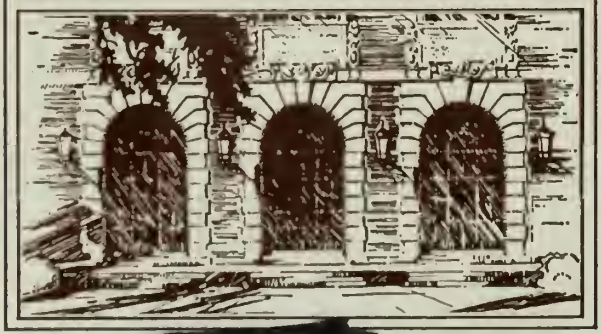
LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

510.84
I16r
no.374-378
cop.2



The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

To renew call Telephone Center, 333-8400

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

DEC 29 1980

APR 4 RECD

L161—O-1096

Digitized by the Internet Archive
in 2013

http://archive.org/details/treetransformati376maru

.84
n Report No. 376
376
p 2

A TREE TRANSFORMATION SYSTEM BASED ON NUCLEOL

by

Kiyoshi Maruyama

FEB 1 9

February, 1970

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

/ of Illinois
FEB 10 1970
Library

The person charging this material is responsible for its return on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

University of Illinois Library

APR - 1 1970

L161—O·1096

Report No. 376

A TREE TRANSFORMATION SYSTEM BASED ON NUCLEOL

by

Kiyoshi Maruyama

February, 1970

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

This work was supported in part by the Department of Computer Science at the University of Illinois at Urbana-Champaign, Urbana, Illinois; the National Science Foundation under NSF Grant GJ-217; by BUILD, the project which is sponsoring cooperation on EOL-4 between the University of Illinois and the University of Colorado; and submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, February, 1970.

ACKNOWLEDGMENT

I would like to express my appreciation to my supervisor, Dr. Jurg Nievergelt, Associate Professor of Computer Science and Mathematics, for his endless guidance and supervision of this thesis.

I also acknowledge the assistance of Mrs. Nurit Bracha, graduate student in Computer Science. I would like to acknowledge the personnel of the IBM 360/75 service area of the Department of Computer Science at the University of Illinois at Urbana-Champaign for their assistance.

This work was supported by the Department of Computer Science at the University of Illinois, by the National Science Foundation under Grant, and by BUILD, the project which sponsors cooperation between the Universities of Illinois and Colorado.

TABLE OF CONTENTS

Page

1. INTRODUCTION 1

2. TREE PROPERTIES AND DEFINITIONS 2

2.1 Rooted ordered trees 2

2.2 Node ordering and parenthesized tree expressions 4

2.3 Definitions 5

3. TREE TRANSFORMATIONS 8

3.1 Tree transformation rule 8

3.2 Pattern matching and replacement 10

3.3 Examples 12

4. TREE TRANSFORMATION ALGORITHMS 18

4.1 Simple tree transformations 18

4.2 Generalized tree transformations 20

5. CONCLUSION 22

LIST OF REFERENCES 23

APPENDICES

A. NUCLEOL PROGRAMS FOR THE TREE TRANSFORMATION ALGORITHMS . . 24

B. NUCLEOL AND TREE ALGORITHMS 41

1. INTRODUCTION

Processing of tree-structured data is of importance in various areas. In particular recent emphasis on transformational grammars for linguistic analysis has raised interest in programming languages where tree transformations can be specified in terms of very general pattern matching and replacement rules.

This thesis presents a formal definition of a very general class of tree transformations, and describes the implementation of this system in terms of the list processing language NUCLEOL [1].

The plan of this thesis is as follows. In Chapter 2, I introduce some properties of trees and definitions. In Chapter 3, I define the class of tree transformations studied in this thesis, and the pattern matching and replacement process which it implies. In Chapter 4 I introduce two systems which are special cases of the class considered: a simple tree transformation and a generalized tree transformation, whose implementation is described in Appendix A. Appendix B contains an introduction to the NUCLEOL language, with examples of programs for a number of standard tree algorithms.

2. TREE PROPERTIES AND DEFINITIONS

Tree structures have been the object of extensive mathematical investigations for many years, long before the advent of computers, and many interesting facts have been discovered.

In this chapter I introduce some definitions and properties of trees which not only give more insight into the nature of tree structures but also have important applications to computer algorithms.

2.1 Rooted ordered trees

In this thesis, "tree" will always mean "rooted ordered tree", since these trees are of greatest interest in linguistic analysis and many other computer applications.

We define a tree T formally as a finite set V of one or more nodes (or vertices) such that

i) There is one specially designated node called the root of the tree, r(T); and

ii) The remaining nodes are partitioned into $m(>0)$ disjoint ordered sets $T_1, T_2, \ldots, T_m$ and each of these sets in turn is a tree. The trees $T_1, \ldots, T_m$ are called the subtrees of r(T). Their roots $r(T_1)$ $r(T_1), \ldots, r(T_m)$ are called the sons (or direct descendants) of r(T), and r(T) is called their father (or direct ancestor). We define a node v itself as a descendant. $r(T_1)$ and $r(T_m)$ are said to be the leftmost and rightmost sons of r(T), respectively. A direct ancestor v of a node w is said to directly dominate w. A node v_1 is an ancestor of (or dominates) a node v_k if there is a sequence $v_1, v_2, \ldots, v_k$ such that v_i directly dominates v_{i+1} for $i = 1, 2, \ldots, k-1$.

A tree S is a subtree of a tree T if and only if either it is a subtree of the root r(T) or else it is a subtree of some subtree of the

root r(T).

The definition of tree just given is recursive, i.e., we have defined a tree in terms of trees. Of course, there are nonrecursive ways to define trees, but a recursive definition seems most appropriate since recursion is a frequently used characteristic of tree structures.

The number of subtrees of a node v is called the degree of that node, denoted by d(v). A node of degree 0 is called a leaf (or terminal node). The level (or depth) of a node v, denoted by $\ell(v)$, in a tree T is defined as follows: the level of the root of a tree T, $\ell(r(T))$, is 0, and the level of any other node v is 1 plus the level of its direct ancestor.

A node v of a tree T is called the deepest common ancestor of nodes $v_1, v_2, \ldots, v_k$ of T if and only if v dominates $v_1, \ldots, v_k$, and the level of v is the highest in the set of levels of all possible common ancestors of $v_1, v_2, \ldots, v_k$. It is obvious that the deepest common ancestor is unique.

In addition to single trees it is useful to consider ordered sets of trees, such as the set of subtrees of the root of a tree. We define a forest to be an ordered set of (zero or more) disjoint trees.

In applications, nodes usually contain some information, which we call the label of a node. Labels are character-strings over some alphabet which need not be specified here. A tree with labeled nodes is called a labeled tree.

Two trees T and S are isomorphic if and only if they have an equal number $n > 0$ of subtrees T_j of r(T) and S_j of r(S), and in addition T_j and S_j are isomorphic for $j = 1, \ldots, n$.

Two labeled trees are isomorphic if and only if they are isomorphic as unlabeled trees, and in addition corresponding nodes have the same label.

A (labeled) tree S is an initial part of tree T if either S is a single node (and, in addition, label of r(S) = label of r(T)) or else the

number of subtrees $S_1, S_2, \ldots, S_m$ of the root r(S) is equal to the number of subtrees $T_1, T_2, \ldots, T_m$ of the root r(T), and S_i is an initial part of T_i for $i = 1,2,\ldots,m$.

2.2 Node ordering and parenthesized tree expressions

It is frequently necessary to enumerate all the nodes of a tree in a linear order, or equivalently, to "traverse" a tree systematically so that each node is "visited" exactly once.

Many algorithms are facilitated if we can talk about the next node following or preceding a given node in such a sequence. Here we consider only one method of enumeration, known as traversing a tree in prefix order (other orders are described in Appendix B).

It is convenient to define prefix order more generally for a forest as well as for a single tree as follows.

To traverse a forest in prefix order: if it is empty, then terminate; otherwise, traverse its first tree in prefix order, and thereafter traverse the forest of the remaining trees in prefix order.

To traverse a tree in prefix order: visit the root, and thereafter traverse the forest of its subtrees in prefix order.

The following sequence is an example such that nodes of the tree in Figure 2.1 are enumerated in prefix order.

A B C D E F G H I

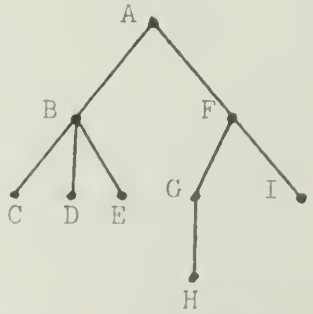


Fig. 2.1 A Tree T

Even if every node has a unique label in a tree, the sequence of these labels in prefix order does clearly not determine the tree uniquely; some additional information about the degree of each node is necessary to convert a sequence of node labels in prefix order into an expression which uniquely represent a labeled tree. One way to achieve this is to add, for every node which is not a leaf, a pair of parentheses which embraces the sequence of labels of all the subtrees of this node. The result of doing this is a parenthesized prefix expression, which determines a unique labeled tree.

The following sequence is the parenthesized prefix expression of the tree given in Figure 2.1.

```
A  B                F  G
(  (  C  D  E  )  (  (  H  )  I  )  )
```

2.3 Definitions

In this section I define some new terms which will be useful for defining tree transformations in later chapters.

Definition 1 (rightmost and leftmost descendant)

A node v is a rightmost (leftmost) descendant of a node w if and only if v is a rightmost (leftmost) son of w or is a rightmost (leftmost) descendant of the rightmost (leftmost) son of w.

Definition 2 (contiguity)

A node v is contiguous to a node w if and only if there are node v' and w' such that:

i) w' is the next brother of v' ,

ii) v is a rightmost descendant of v' and

iii) w is a leftmost descendant of w' .

Definition 3 (contiguous sequence)

A sequence of nodes: $v_1, v_2, \ldots, v_k$ is said to be a contiguous sequence if and only if v_i is contiguous to v_{i+1}, for $i = 1,2,\ldots,k-1$.

For example, a sequence C D E K L J is a contiguous sequence of tree in Figure 2.2.

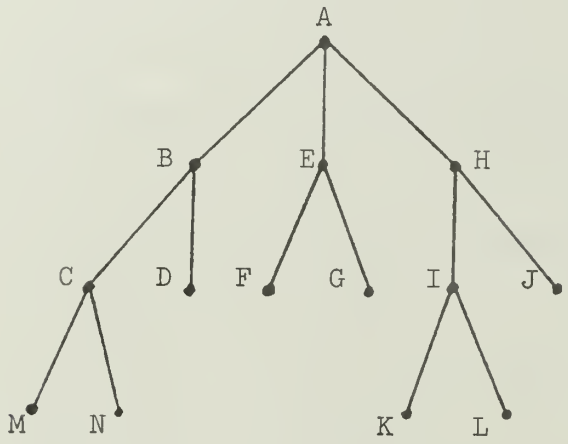


Fig. 2.2 Tree T

We now extend some of the relations defined earlier among nodes to hold between subtrees as well as their initial parts.

Definition 4 (dominance and direct dominance)

Let two subtrees (or initial parts) of a tree T be T_1 and T_2, then T_1 (directly) dominates T_2 if and only if $r(T_1)$ (directly) dominates $r(T_2)$.

Definition 5 (contiguity)

Let two subtrees (or initial parts) of a tree T be T_1 and T_2, then T_1 is contiguous to T_2 if and only if $r(T_1)$ is contiguous to $r(T_2)$.

Definition 6 (contiguous sequence)

A sequence $T_1, T_2, \ldots, T_k$ of subtrees (or initial parts) of a tree T

is said to be a contiguous sequence if and only if $r(T_i)$ is contiguous to $r(T_{i+1})$, for $i = 1,2,\ldots,k-1$.

3. TREE TRANSFORMATIONS

The tree transformations considered here consist of a single rule of the form "p → r", where p is a pattern and r is a replacement. The rule is applied to a tree T by matching p against T; if this match is successful then the matching portion of T is replaced by a tree which is defined by r and the matching portion of T.

In linguistic analysis and other applications, one is, of course, interested in more elaborate tree transformations, which consist of several rules and additional comments for specifying the sequence in which individual rules are to be applied. This thesis is not concerned with the latter aspect. The special tree transformations implemented here, however, form part of a larger system in which the control junctions (sequencing of rules, I/O, etc.) are programmed in PL/1 (see [2]).

3.1 Tree transformation rule

The following is the most important formal definition of tree transformation rule, in which pattern and replacement are defined recursively.

<rule> ::= <pattern> → <replacement>

<pattern> ::= <p-element> | <p-element> (<pattern>*) | <r>

<replacement> ::= <r-element> | <r-element> (<replacement>*)

<p-element> ::= <pea> | (<pea>*) | any

<pea> ::= <p-name> <attribute>

<p-name> ::= <name> | <tree variable> | *

<attribute> ::= <null> | <level> | <degree> | <level> <degree>

<r-element> ::= <name> | <tree variable> | # | @

<name> ::= <characterstring>

<r> ::= <relation>

At this moment <relation>, <level>, <degree>, <tree variable> and <characterstring> are not specified. They will be discussed in later sections in specific cases, but their function in tree transformations can roughly be described as follows: <relation> allows the specification of a relation which must hold between the parts of a tree which are matched against the patterns listed between the parentheses. Examples of relations which will be considered are direct dominance and contiguity, but we do not now want to restrict the possible relations which one might consider. <tree variable> allows the naming of matched subtrees of the object tree so they can be used in the replacement part. <level> and <degree> represent additional constraints which may be imposed (optionally) on matching nodes. Finally, <characterstring> is unspecified in order not to decide on a particular character set.

The grammar just given assigns to any pattern and any replacement a tree (the conventional "parsing tree"), which we call the pattern tree and replacement tree, respectively.

A rule is said to be well-defined if and only if the replacement tree is an initial part of the pattern tree (disregarding all node labels); otherwise it is ill-defined.

The following two examples show a well-defined rule and an ill-defined rule.

Example 1 (well-defined rule).

<p-element> → (<r> <p-element>) <p-element> → (<r> <p-element> <p-element>))

<r-element> → (<r-element>) <r-element> → (<r-element> <r-element>))

Example 2 (ill-defined rule).

<p-element> → (<r> <p-element> <p-element> <p-element>)

<r-element> → (<r-element> <r-element>)

Hereafter, we always assume that given rules are well defined.

3.2 Pattern matching and replacement

The process of applying a transformation rule R: p → r to a given tree T (called the object tree) proceeds in two steps: First, (the matching process), p is matched against T in an attempt to find a subtree S of T (the matched subtree). If this matching attempt is successful, we say R is applicable to T, and we continue with the replacement process; if the match is unsuccessful, we say R is inapplicable to T, and the process stops without altering T.

Second (the replacement process), the matched subtree S of T is replaced by a subtree S' which is determined by both the replacement part r of the rule and by the matched subtree S.

These two processes will now be described in detail.

Matching process

The matched subtree S is the first subtree of T in prefix order, of which the pattern tree p is an initial part, subject to the following conditions:

1) The level and degree of every node w of S must satisfy the requirements, if any, stated in the <attribute> of the corresponding node v of the pattern tree p.

2) The label of every node w of S must be equal to the level of the corresponding node v of p, except if v is the special <p-element>, * (which "matches every node").

3) The special <p-element> of the form (<pea>*) is considered to be a single node in the pattern tree p, which matches the corresponding node w of S if at least one of the <pea>'s contained in it does.

4) A <tree variable> in the pattern tree p assumes as its value the subtree of T to whose root it matches.

Replacement process

Each node in the matched subtree S is replaced by the unique <r-element> (of the replacement r) which corresponds to the <p-element> (of the pattern p) to which it matched. In this replacement, the special <r-elements> #, ∂, and <tree variable> are treated as follows:

1) # : no replacement is done.

2) ∂ : delete corresponding <p-element> as well as the subtree whose dominant is <p-element>.

3) <tree variable> : replace by the subtree which was assigned to this <tree variable> during the matching process.

3.3 Examples

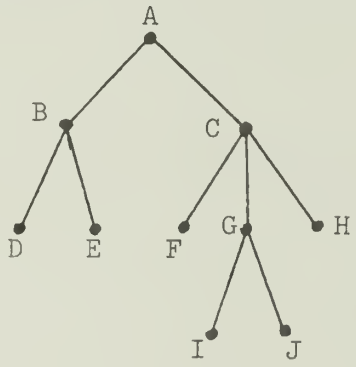
The following examples of tree transformations are presented to illustrate the previous definitions.

Notation and restrictions:

<characterstring>'s are denoted by single capital letters (A, B, C, ...). <level> is restricted to level 0 and denoted by a subscript "r" (for root). <degree> is restricted to degree 0 and denoted by a subscript "t" (for terminal node). Tree variables are denoted by T_1, T_2, ... <relations>'s are restricted to two: direct dominance, denoted by "d", and contiguity, denoted by "c".

R(T) denotes the result of applying rule R to tree T.

In each example, the tree of Figure 2.3(a) will be used as the object tree. Its corresponding parenthesized prefix expression is shown in Figure 2.3(b).



(a) Tree T

A B C G
((D E) (F (I J) H))

(b) Parenthesized prefix tree expression of T

Fig. 2.3 The object tree

Example 1. $R_1(T)$

$$R_1 : \overset{C}{\underset{d}{(}} * \; T_1 \; H_t \;) \rightarrow \overset{\#\; F}{(} \; (\; T_1 \;) \; \partial \; \overset{H}{(} \; T_1 \; F \;) \;)$$

$$R_1(T) : \overset{A\; B}{(} \; (\; D \; E \;) \; \overset{C\; F\; G}{(} \; (\; (\; I \; J \;) \;) \; \overset{H\; G}{(} \; (\; I \; J \;) \; F \;) \;) \;)$$

where $T_1 : \overset{G}{(} \; I \; J \;)$

$* : F$

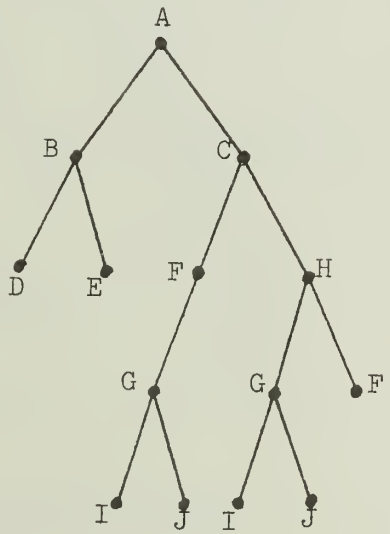


Fig. 2.4 The tree of $R_1(T)$

Example 2. $R_2(T)$

$$R_2 : \overset{*}{\underset{d}{(}}\ \overset{B}{(}\ D_t\ E\ )\ \overset{C}{\underset{d\ any}{(}}\ (\ K\ F\ )\ T_1\ *\ )\ )$$

$$\rightarrow \overset{\#\ \#\ D}{(}\ (\ T_1\ E\ )\ \hat{a}\ )\ \overset{G}{(}\ \#\ \#\ \hat{a}\ )\ )$$

$$R_2(T) : \overset{A\ B\ D\ G}{(}\ (\ (\ I\ J\ )\ E\ )\ )\ \overset{G\quad G}{(}\ \mathbf{F}\ (\ I\ J\ )\ )\ )$$

where $T_1 : \overset{G}{(}\ I\ J\ )$

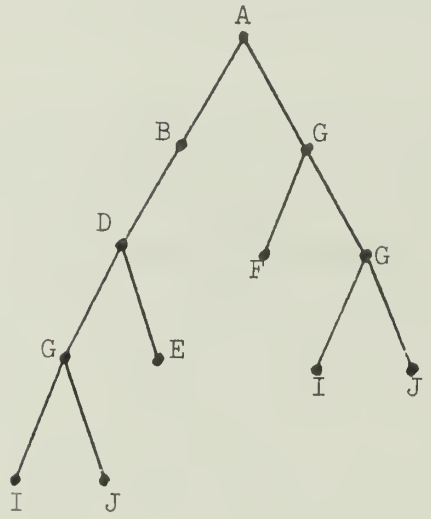


Fig. 2.5 The tree of $R_2(T)$

Example 3. $R_3(T)$

$$R_3 : \overset{*}{\underset{c}{(}}\ B\ F_t\ G\ H\ ) \rightarrow \overset{S}{(}\ \#\ \#\ \partial\ G\ )$$

$$R_3(T) : \overset{S}{(}\ \overset{B}{(}\ D\ E\ )\ \overset{C}{(}\ F\ G\ )\ )$$

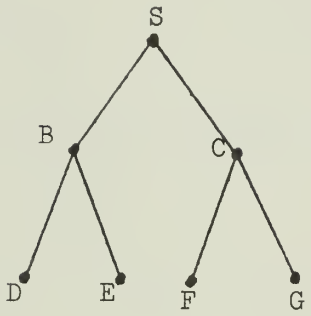


Fig. 2.6 The tree of $R_3(T)$

Example 4. $R_4(T)$

$$R_4 : \overset{A}{\underset{C}{(}}\ B\ F_t\ \overset{G}{\underset{d}{(}}\ I_t\ T_1\ )\ T_2\ )$$

$$\rightarrow \overset{S}{(}\ \#\ T_1\ \overset{\#}{(}\ K\ T_2\ )\ T_1\ )$$

$$R_4(T) : \overset{S}{(}\ \overset{B}{(}\ D\ E\ )\ \overset{C}{(}\ J\ \overset{G}{(}\ K\ H\ )\ J\ )\ )$$

where T_1 : J $\qquad$ T_2 : H

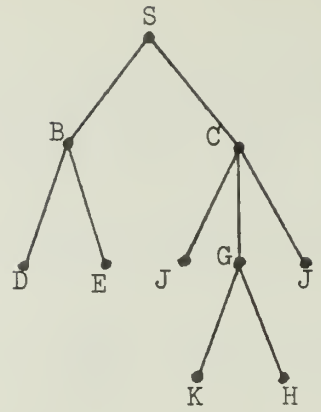


Fig. 2.7 The tree of $R_4(T)$

Example 5. $R_5(T)$

$$R_5 : \overset{*}{\underset{d}{(}}\ T_1\ \overset{C}{\underset{c}{(}}\ F_t\ T_2\ H\ )\ )$$

$$\rightarrow \overset{\#}{(}\ T_2\ \overset{D\ F}{(}\ (\ T_2\ )\ \omega\ T_1\ )\ )$$

$$R_5(T) : \overset{A\ G}{(}\ (\ I\ J\ )\ \overset{D\ F\ G}{(}\ (\ (\ I\ J\ )\ )\ \overset{B}{(}\ D\ E\ )\ )\ )$$

where $T_1 : \overset{B}{(}\ D\ E\ )$, $T_2 : \overset{G}{(}\ I\ J\ )$

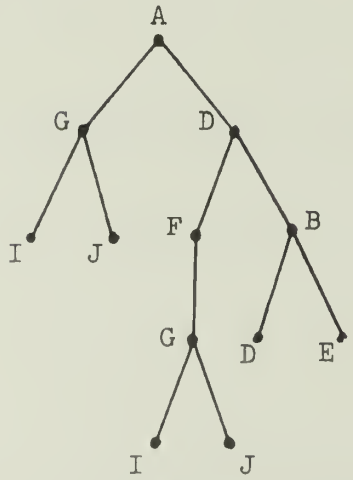


Fig. 2.8 The tree of $R_5(T)$

Example 6. $R_6(T)$

$$R_6 : \overset{A}{\underset{c}{(}}\ D_t\ E\ \overset{*}{\underset{c}{(}}\ F\ G\ H_t\ )\ )$$

$$\rightarrow \overset{\#}{(}\ \#\ \wp\ \overset{S}{(}\ \#\ \wp\ \#\ )\ )$$

$$R_6(T) : \overset{A\ B}{(}\ (\ D\ )\ \overset{S}{(}\ F\ H\ )\ )$$

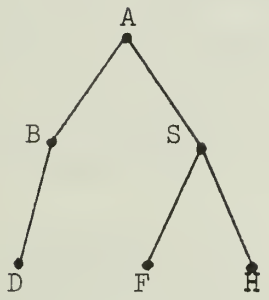


Fig. 2.9 The tree of $R_6(T)$

4. TREE TRANSFORMATION ALGORITHMS

In this chapter I introduce two kinds of tree transformations, called simple and general, respectively. An algorithm for performing the pattern matching and replacement process is described for each case.

4.1 Simple tree transformations

A simple tree transformation is one whose pattern is generated by the single relation "direct dominance" and by <p-elements>'s of type <name> and <tree variable>, only. Formally (see definition of rule in section 3.1), we say that a simple tree transformation is a rule defined by:

<rule> ::= <pattern> → <replacement>

$$\langle \text{pattern} \rangle ::= \langle \text{element} \rangle \;\Big|\; \begin{matrix} \langle \text{element} \rangle \\ (\langle \text{pattern} \rangle *) \\ \langle r \rangle \end{matrix}$$

$$\langle \text{replacement} \rangle ::= \langle \text{element} \rangle \;\Big|\; \begin{matrix} \langle \text{element} \rangle \\ (\langle \text{replacement} \rangle *) \end{matrix}$$

<element> ::= <name> | <tree variable>

<r> ::= <direct dominance>

In other words, both the pattern and the replacement are fixed, explicitly given trees, whose terminal nodes may be tree variables. With these restrictions, there is a simple algorithm for the pattern matching and replacement process.

The skeleton of this simple tree transformation algorithm, called [illegible], is shown in Fig. 4.1, in a notation which shows the relationship among the programs of which it consists.

Each node represents an external procedure (a "wfs" in NUCLEOL terminology), and arrows labeled [illegible] represent transfer of control between them.

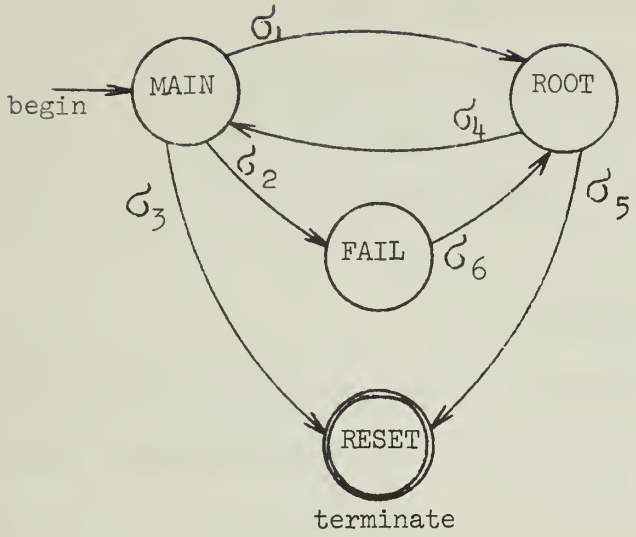


Fig. 4.1 Structure of the STT algorithm

ALGORITHM STT (Simple Tree Transformation)

MAIN. [Initialize the system and carry out the matching and replacement process]

MAIN. 1 Initialize the system, then go unconditionally (σ_1) to ROOT.

MAIN. 2 [Matching process] Match the pattern to the subtree (whose root is determined by ROOT) of the object tree. If the match fails (σ_2), go to FAIL.

MAIN. 3 [Tree variables] Construct a table for tree variables and their corresponding subtrees of the object tree. Using this table replace tree variables by act al subtrees (deletion and attachment of subtrees).

MAIN. 4 [Replacement]
Replace the matched subtree of the object tree by the (modified) pattern.

MAIN. 5 [Termination]
Go to RESET (σ_3).

ROOT [Match root of pattern tree]

Find the next node in the object tree whose label is equal to the label of the root of the pattern tree, and if found (σ_4), go to MAIN. 2 . If no such node is found, then the rule is not applicable, go to RESET (σ_5).

FAIL [Restore object, tree]

Since the pattern did not match the subtree whose root was chosen by ROOT, restore the object tree, then go to ROOT (σ_6).

RESET [Termination]

Reset the system, then terminate.

The NUCLEOL implementation of this algorithm is listed in Appendix A, Algorithm STT.

4.2 General tree transformations

General tree transformations are rules generated by the relations direct dominance and contiguity.

Formally we restate:

$$\text{<pattern>} ::= \text{<p-element>} \mid \begin{matrix}\text{<p-element>}\\ (\\ \text{<r>}\end{matrix} \quad \text{<pattern>} *)$$

$$\text{<replacement>} ::= \text{<r-element>} \mid \begin{matrix}\text{<r-element>}\\ (\end{matrix} \quad \text{<replacement>} *)$$

$$\text{<r>} ::= \text{<direct dominance>} \mid \text{<contiguity>}$$

<p-elements> and <r-element> are as defined in Chapter 3. Notice that one can define new relations between matching subtrees in terms of the two given relations. E.g., the relation "brotherhood" can be defined in terms of direct dominance. The relation of brotherhood between nodes A, B and C, for example, can be expressed as

$$\begin{matrix}\text{<p-element>}\\ (_{d}\end{matrix} \; A \; B \; C \;) \text{ or } \begin{matrix}\text{<p-element>}\\ (_{d}\end{matrix} \; \begin{matrix}A\\ (_{d}\end{matrix} \; T_1 \;) \; \begin{matrix}B\\ (_{d}\end{matrix} \; T_2 \;) \; \begin{matrix}C\\ (_{d}\end{matrix} \; T_3 \;) \;) \; .$$

This general tree transformation algorithm, called GTT, consists of four main parts : i) the main part, which consists of NUCLEOL wfs "MAIN" alone, controls transfers to all other parts, ii) the matching part, which consists of wfs's "MATCH', "ROOT", "D-DOMI", "CONTI", "ANYONE" and "FAIL", checks whether or not the given rule is applicable to the object tree, iii) the replacement part, which consists of wfs's "TRANS", "D-TO" and "C-TO", and iv) the terminal part consisting of wfs "RESET".

The NUCLEOL program which implements general tree transformations is shown in Appendix A, Algorithm GTT.

5. CONCLUSION

Recent emphasis on transformational grammars for linguistic analysis has raised interest in programming languages in which tree transformations can be specified in terms of very general pattern matching and replacement rules.

Our definition of a transformation rule consisting of pattern and replacement (see section 3.1), based recursively on elementary patterns and relations among patterns, appears to be a natural generalization of all the transformation rules which have been used in transformational grammars.

The system described here, which can carry out a single transformation on an object tree, is currently being augmented by a control language (based on PL/1) which allows the specification of sequences of transformations [2].

LIST OF REFERENCES

1. Nievergelt, J., Fischer, F., Irland, M. I., Sidlo, J. R. "NUCLEOL - A Minimal List Processor", Proc. Purdue Centennial Year Symposium on Information Processing, pp. 92-103, 1969. Also, University of Illinois, Department of Computer Science, Report No. 324, April, 1969.

2. Bracha, N. Forthcoming M.S. Thesis, University of Illinois, Department of Computer Science.

APPENDIX A

NUCLEOL PROGRAMS FOR THE TREE TRANSFORMATION ALGORITHMS

ALGORITHM SIT:

```
/***** OBJECT TREE *****/
$S  'TREE'  $(X  $C  'A'  $(  $C  'B'  $(  $C  'D'  $C  'E'  $)
    $C  'C'  $(  $C  'F'  $C  'G'  $)  $)  $)X

/***** PATTERN *****/
$S  'LHS'  $(X       $C  'B'  $(  $C  'D'  $(  $CV  'T1'  $)
             $CV  'T2'  $)  $)X

/***** REPLACEMENT *****/
$S  'RHS'  $(X       $C  'A'  $(  $C  'B'  $(  $CV  'T2'  $)
 $C  'D'      $(  $CV  'T1'  $)  $)  $)X

/***** TEMPORARY STORAGE *****/
$S  'TEMP'  $(X  $)X

/***** TABLE FOR TREE VARIABLE CORRESPONDENCE *****/
$S  'CORRESPO'  $(X  $CV  'TO'  $)X

/***** OPERATION ROOM *****/
$S  'OP_ROOM'  $(X  $)X


/***** REFERENCE FOR MATCHING FAILED *****/
$S  'FAIL3'  $(XN  $CK  'RSTR'  $R  'TEMP'
                   $CK  'SHFT'  $RR  'TEMP'
                   $CK  'MOVE'  $RR  'TEMP'  $RR  'TREE'
                   $CK  'SHFT'  $RR  'TREE'
                   $CK  'RSTR'  $R  'LHS'
                   $CK  'SHFT'  $RR  'LHS'
                   $(N  $CK  'TEST'  $RL  'CORRESPO'
                            $C  'A =D'  $B  '1'
                        $(FU  $CK  'MOVE'  $RL  'CORRESPO'
                                   $R  'SYS_FREE'
                          $)N
                   $)S
                   $R  'ROOT'
             $)X


/***** REFERENCE FOR RESETTING PROGRAM *****/
$S  'RESET'  $(XN  $CK  'RSTR'  $R  'TREE'
                   $CK  'RSTR'  $R  'LHS'
                   $CK  'RSTR'  $R  'RHS'
                   $CK  'RSTR'  $R  'TEMP'
                   $CK  'RSTR'  $R  'CORRESPO'
                   $CK  'RSTR'  $R  'OP_ROOM'
                   $R  'SYS_STOP'
             $)X
```

```
/***** REFERENCE FOR MATCHING OF THE ROOT OF THE TREE *****/
$S   'ROOT'   $(XN
$(N     $CK   'TEST'   $RR   'TREE'   $C   '   = '   $RR   'LHS'
      $(FU   $CK   'SHFT'   $RR   'TREE'
             $CK   'TEST'   $RR   'TREE'   $C   'A =D'   $B   '1'
             $(S   $R   'RESET'   $)
      $)FU
$)S
      $CK   'SHFT'   $RR   'TREE'
      $CK   'SHFT'   $RR   'LHS'
      $CK   'TEST'   $RR   'TREE'   $C   '   = '   $RR   'LHS'
      $(S   $CK   'MOVE'   $RR   'TREE'   $RR   'TEMP'
            $RR   'MATCH'
      $)
      $(FU   $CK   'SHFT'   $RL   'LHS'   $)N
      $R   'ROOT'
$)X


/***** MAIN PROGRAM *****/
$SN   'MAIN'   $(XN
      $CK   'SHFT'   $RR   'TEMP'
      $CK   'SHFT'   $RR   'CORRESPO'
          $CK   'SHFT'   $RR   'OP_ROOM'
      $CK   'SHFT'   $RR   'TREE'
      $CK   'SHFT'   $RR   'LHS'
      $CK   'SHFT'   $RR   'RHS'
$R   'ROOT'
$(N   $CK   'TEST'   $RR   'LHS'   $C   'A =D'   $C   'V'
      $(S   $CK   'COPY'   $RR   'LHS'   $RL   'CORRESPO'
            $CK   'SHFT'   $RR   'LHS'
            $(N   $CK   'TEST'   $RR   'LHS'   $C   '   = '   $RR   'TEMP'
                  $(FU   $CK   'COPY'   $RR   'TEMP'   $RL   'CORRESPO'
                         $CK   'MOVE'   $RR   'TEMP'   $RL   'TEMP'
                  $)N
            $)S
            $CK   'SHFT'   $RR   'LHS'
            $CK   'SHFT'   $RR   'TEMP'
      $)N
      $(FU   $CK   'TEST'   $RR   'TEMP'   $C   '   = '   $RR   'LHS'
            $(FU   $CK   'TEST'   $RR   'LHS'   $C   'T =D'   $C   '('
            $(S    $CK   'SHFT'   $RR   'LHS'
                   $CK   'TEST'   $RR   'LHS'   $C   'A =D'   $C   'V'
                   $(FU   $R   'FAIL3'   $)
                   $CK   'COPY'   $RR   'LHS'   $RL   'CORRESPO'
                   $CK   'SHFT'   $RR   'LHS'
                   $CK   'TEST'   $RR   'LHS'   $C   'T =D'   $C   ')'
                   $(FU   $R   'FAIL3'   $)
                   $CK   'SHFT'   $RR   'LHS'
            $)S
            $(FU   $R   'FAIL3'   $)
            $)N
            $(S   $CK   'SHFT'   $RR   'TEMP'
```

```
                $CK  'SHFT'  $RR  'LHS'
          $)N
     $)S
     $CK  'TEST'  $RR  'TEMP'  $C  'A =D'  $B  '1'
$)S
     $CK  'TEST'  $RR  'LHS'  $C  'A =D'  $B  '1'
     $(FU  $R  'FAIL3'  $)
    $CK  'SHFT'  $RR  'RHS'
    $CK  'TEST'  $RR  'RHS'  $C  'T =D'  $C  '('
    $(S  $CK  'COPY'  $RR  'RHS'  $RR  'OP_ROOM'  $)N
    $CK  'COPY'  $RL  'RHS'  $RR  'OP_ROOM'
    $(NFU
         $CK  'TEST'  $RR  'OP_ROOM'  $C  'A =D'  $C  'V'
          $(S  $CK  'RSTR'  $RL  'CORRESPO'
              $(N  $CK  'TEST'  $RR  'CORRESPO'  $C  ' = '
                                     $RR  'OP_ROOM'
                   $(FU  $CK  'MOVE'  $RR  'CORRESPO'
                                      $RL  'CORRESPO'
                         $CK  'TEST'  $RR  'CORRESPO'
                              $C  'A =D'  $C  'V'
                   $)S
              $)S
              $CK  'SHFT'  $RR  'CORRESPO'
              $CK  'MOVE'  $RR  'OP_ROOM'  $R  'SYS_FREE'
              $(N  $CK  'TEST'  $RR  'CORRESPO'  $C  'A =D'
                                     $C  'V'
                   $(FU  $CK  'COPY'  $RR  'CORRESPO'
                                      $RL  'OP_ROOM'
                         $CK  'MOVE'  $RR  'CORRESPO'
                                      $RL  'CORRESPO'
                   $)N
              $)S
/**  TREE  VARIABLE  IS  NULL  **/
              $CK  'TEST'  $RL  'OP_ROOM'  $C  'T =D'  $C  '('
              $(S  $CK  'TEST'  $RR  'OP_ROOM'  $C  'T =D'
                                     $C  ')'
                   $(S  $CK  'SHFT'  $RR  'OP_ROOM'
                        $CK  'MOVE'  $RL  'OP_ROOM'
                                     $R  'SYS_FREE'
                   $)N
              $)N
              $CK  'SHFT'  $RL  'OP_ROOM'
         $)NFU
         $CK  'SHFT'  $RR  'OP_ROOM'
         $CK  'TEST'  $RR  'OP_ROOM'  $C  'A =D'  $B  '1'
    $)S
    $CK  'MOVE'  $RL  'TREE'  $R  'SYS_FREE'
    $(N  $CK  'TEST'  $RL  'OP_ROOM'  $C  'T =D'  $C  '('
         $(FU  $CK  'MOVE'  $RL  'OP_ROOM'  $RR  'TREE'  $)N
    $)S
    $CK  'MOVE'  $RL  'TEMP'  $R  'SYS_FREE'
    $CK  'RSTR'  $RL  'CORRESPO'
    $(N  $CK  'TEST'  $RR  'CORRESPO'  $C  '  = '  $CV  'TO'
```

```
        $(FU  $CK  'MOVE' $RR  'CORRESPO'  $R  'SYS_FREE'  $)N
    $)S
    $CK  'COPY'  $R  'TREE'  $R  'SYSPRINT'
    $R  'RESET'
$)
$)X
```

ALGORITHM GTT:

```
/***** TREE OBJECT *****/
$S  'TREE'  $(X  $CR  'A'  $(  $C  'B'  $(  $CT  'D'
    $CT  'E'  $)  $C  'C'  $(  $CT  'F'  $C  'G'  $(  $CT  'I'
    $CT  'J'  $)  $CT  'H'  $)  $)  $)X

/***** PATTERN *****/
$S  'PATTERN'
  $(X  $C  'A'  $(  $CV  'T1'  $C  'C'  $(S  $CT  'F'  $C  'G'
    $(  $CV  'T2'  $)  $C  'H'  $)  $)  $)X

/***** REPLACEMENT *****/
$S  'REPLACE'
$(X  $C  'S'  $(  $CV  'T1'  $C  'D'  $(S  $C  'F'
    $(  $CV  'T2'  $)  $C  '#'  $C  '@'  $)  $)  $)X

/***** TEMPORARY MEMORY FOR PATTERN *****/
$S  'LHS'  $(X  $)X

/***** TEMPORARY MEMORY FOR REPLACEMENT *****/
$S 'RHS'  $(X  $)X

/***** OPERATION ROOM FOR TREE *****/
$S  'OP_ROOM'  $(X  $)X

/***** CORRESPONDENCE TABLE *****/
$S  'CORRESPO'  $(X  $CV  'TO'  $)X

/***** TEMPORARY STORAGE *****/
$S  'TEMP'  $(X  $)X


/***** REFERENCE FOR BACKING UP SCANNER TO THE ORIGINAL
                                    POSITION *****/
$S  'SET_SCAN'  $(XN  $(N  $CK  'TEST'  $RR  'LHS'  $C  '  = '
                                    $C  'SCAN'
                        $(FU  $CK  'MOVE'  $RR  'LHS'
                                    $RL  'LHS'  $)N
                    $)S
                    $CK  'MOVE'  $RR  'LHS'  $R  'SYS_FREE'
                    $RR  'MATCH'
                $)X


/***** REFERENCE FOR RESETTING EACH WFS AND TERMINATION *****/
$S  'RESET'  $(XN  $CK  'RSTR'  $R  'TREE'
                   $CK  'RSTR'  $R  'PATTERN'
                   $CK  'RSTR'  $R  'LHS'
                   $CK  'RSTR'  $R  'REPLACE'
                   $CK  'RSTR'  $R  'RHS'
```

```
                $CK   'RSTR'   $R   'OP_ROOM'
                $CK   'RSTR'   $R   'CORRESPO'
                $CK   'RSTR'   $R   'TEMP'
                $R   'SYS_STOP'
        $)X


/***** REFERENCE FOR MATCHING FAILED *****/
$S   'FAIL'   $(XN   $CK   'RSTR'   $R   'LHS'
                     $CK   'SHFT'   $RR   'LHS'
                     $(N   $CK   'TEST'   $RR   'LHS'   $C   'A =D'
                                                $B   '1'
                           $(FU   $CK   'MOVE'   $RR   'LHS'
                                                $R   'SYS_FREE'   $)N
                     $)S
                     $CK   'RSTR'   $R   'OP_ROOM'
                     $CK   'SHFT'   $RR   'OP_ROOM'
                     $(N   $CK   'TEST'   $RR   'OP_ROOM'   $C   'A =D'
                                                $B   '1'
                           $(FU   $CK 'MOVE' $RR 'OP_ROOM'
                                                $R   'SYS_FREE'   $)N
                     $)S
                     $(N   $CK   'TEST'   $RL   'CORRESPO'
                                                $C   'A =D'   $B   '1'
                           $(FU   $CK   'MOVE'   $RL   'CORRESPO'
                                                $R   'SYS_FREE'   $)N
                     $)S
                     $R   'MATCH'
        $)X


/***** ROOT DOES MATCHING FOR THE ROOT OF A TREE *****/
$S   'ROOT'   $(XN
     $CK   'TEST'   $RR   'LHS'   $C   'T =D'   $C   'C'
     $(S   $CK   'TEST'   $RR   'LHS'   $C   '   = '   $C   '*'
           $(FU   $CK   'TEST'   $RR   'LHS'   $C   '   = '   $RR   'TREE'
                  $(FU   $CK   'SHFT'   $RR   'TREE'
                         $CK   'TEST'   $RR   'TREE'   $C   'A =D'
                                                $B   '1'
                         $(S   $R   'RESET'   $)
                  $)FU
           $)S
           $CK   'TEST'   $RR   'TREE'   $C   'A =D'   $C   'T'
           $(S   $CK   'SHFT'   $RR   'TREE'
                 $CK   'TEST'   $RR   'TREE'   $C   'A =D'   $B   '1'
                 $(S   $R   'RESET'   $)
                 $(FU   $CK   'TEST'   $RR   'TREE'   $C   'T =D'
                                                $C   'C'
                        $(S   $R   'ROOT'   $)
                 $)FU
           $)
     $)NFU
     /** SHOULD BE ANYONE **/
```

```
    $(FU  $CK  'SHFT'  $RR  'LHS'
    $(N  $(NS  $CK  'TEST'  $RR  'TREE'  $C  '  = '  $RR  'LHS'
               $(FU  $CK  'SHFT'  $RR  'LHS'
                     $CK  'TEST'  $RR  'LHS'  $C  'T =D'
                                      $C  ')'
                     $(S  $CK  'RSTR'  $RL  'LHS'
                          $(N  $CK  'SHFT'  $RR  'TREE'
                               $CK  'TEST'  $RR  'TREE'
                                    $C  'T =D'  $C  'C'
                               $(FU  $CK  'TEST'  $RR  'TREE'
                                    $C  'A =D'  $B  '1'
                                  $(S  $R  'RESET'  $)
                               $)FU
                          $)S
                     $)N
               $)FUN
          $)S
          $CK  'TEST'  $RR  'TREE'  $C  'A =D'  $C  'T'
    $)FU
          $CK  'MOVE'  $RR  'LHS'  $RR  'TEMP'
          $CK  'RSTR'  $RR  'LHS'
          $CK  'SHFT'  $RR  'LHS'
          $CK  'MOVE'  $RL  'LHS'  $R  'SYS_FREE'
          $CK  'MOVE'  $RR  'TEMP'  $RR  'LHS'
    $)N
    $CK  'SHFT'  $RR  'LHS'
    $CK  'SHFT'  $RR  'TREE'
    $CK  'COPY'  $RR  'TREE'  $RR  'OP_ROOM'
    $CK  'COPY'  $RL  'TREE'  $RL  'OP_ROOM'
    $RR  'MATCH'
$)X


/***** REFERENCE FOR ANY ONE OF THE SPECIFIED NODES *****/
$S  'ANYONE'  $(XN  $CK  'SHFT'  $RR  'LHS'
    $(N  $CK  'TEST'  $RR  'LHS'  $C  '  = '  $RR  'OP_ROOM'
         $(FU  $CK  'SHFT'  $RR  'LHS'
               $CK  'TEST'  $RR  'LHS'  $C  'T =D'  $C  ')'
               $(S  $R  'FAIL'  $)
         $)FU
    $)S
    $CK  'TEST'  $RR  'LHS'  $C  'A =D'  $B  '0'
    $(FU  $CK  'TEST'  $RR  'LHS'  $C  'A =A'  $RR  'OP_ROOM'
          $(FU  $R  'ANYONE'  $)
    $)S
    $CK  'MOVE'  $RR  'LHS'  $RR  'TEMP'
    $CK  'RSTR'  $RR  'LHS'
    $CK  'SHFT'  $RR  'LHS'
    $CK  'MOVE'  $RL  'LHS'  $R  'SYS_FREE'
    $CK  'MOVE'  $RR  'TEMP'  $RL  'LHS'
    $CK  'SHFT'  $RR  'OP_ROOM'
    $RR  'MATCH'
$)X
```

```
/***** REFERENCE FOR DIRECT DOMINATION *****/
$S  'D_DOMINE'  $(XN
    $CK  'TEST'  $RR  'LHS'  $C  'T =D'  $C  '('
    $(S  $CK  'TEST'  $RR  'LHS'  $C  '  = '  $RR  'OP_ROOM'
              $(FU $CK  'SHFT'  $RR  'LHS'
                   $CK 'TEST' $RR 'LHS' $C 'A =D' $C 'V'
                   $(FU  $R  'FAIL'  $)
                   $CK  'COPY'  $RR  'LHS'  $RL  'CORRESPO'
                   $CK  'SHFT'  $RR  'LHS'
                   $CK 'TEST' $RR 'LHS' $C 'T =D' $C ')'
                   $(FU  $R  'FAIL'  $)
                   $CK  'SHFT'  $RR  'LHS'
                  $CK  'MOVE'  $RL  'LHS'  $R  'SYS_FREE'
                   $RR  'MATCH'
              $)
    $)SN
    $(FU  $CK  'TEST'  $RR  'LHS'  $C  '  = '  $RR  'OP_ROOM'
          $(S  $CK  'TEST'  $RR  'LHS'  $C  'A =D'  $B  'O'
               $(FU  $CK  'TEST'  $RR  'LHS'  $C  'A =A'
                                  $RR  'OP_ROOM'
                     $(FU  $R  'FAIL'  $)
               $)S
         $)SN
         $(FU  $CK  'TEST'  $RR  'LHS'  $C  'A =D'  $C  'V'
               $(S  $CK  'COPY'  $RR  'LHS'  $RL  'CORRESPO'
                    $CK  'SHFT'  $RR  'LHS'
                    $(N  $CK  'TEST'  $RR  'LHS'  $C  '  = '
                                      $RR  'OP_ROOM'
                         $(FU  $CK  'MOVE'  $RR  'OP_ROOM'
                                      $RL  'CORRESPO'  $)N
                    $)S
               $)N
               $(FU  $CK  'TEST'  $RR  'LHS'  $C  '  = '
                                      $C  '*'
                    $(S  $CK  'MOVE'  $RR  'LHS'
                                      $R  'SYS_FREE'
                         $(N  $CK 'TEST' $RR 'LHS' $C '  = '
                                      $RR  'OP_ROOM'
                              $(FU  $CK  'COPY' $RR 'OP_ROOM'
                                      $RL  'LHS'
                                    $CK 'MOVE' $RR 'OP_ROOM'
                                      $RL  'OP_ROOM'
                              $)N
                         $)S
                         $CK  'SHFT'  $RL  'LHS'
                         $CK  'SHFT'  $RL  'OP_ROOM'
                     $)N
                     $(FU  $R  'FAIL'  $)
               $)N
         $)N
    $)N
    $CK  'SHFT'  $RR  'LHS'
    $CK  'SHFT'  $RR  'OP_ROOM'
```

```
    $CK  'TEST'  $RR  'LHS'  $C  'T =D'  $C  '('
    $(S  $RR  'MATCH'  $)
    $CK  'TEST'  $RR  'LHS'  $C  'T =D'  $C  ')'
    $(S  $CK  'TEST'  $RR  'LHS'  $C  'A =D'  $B  '1'
         $(S  $RR  'MATCH'  $)
         $CK  'TEST'  $RR  'LHS'  $C  '  = '  $RR  'OP ROOM'
         $(S  $CK  'SHFT'  $RR  'LHS'  $CK  'SHFT'
                                       $RR  'OP_ROOM'  $)N
         $RR  'MATCH'
    $)
    $R  'D_DOMINE'
$)X


/***** REFERENCE FOR CONTIGUITY *****/
$S  'CONTI'  $(XN
    $CK  'TEST'  $RR  'LHS'  $C  'T =D'  $C  'C'
    $(S  $CK  'TEST'  $RR  'OP_ROOM'  $C  'T =D'  $C  '('
         $(S  $CK  'MOVE'  $RR  'OP_ROOM'  $RL  'OP_ROOM'  $)N
         $CK  'TEST'  $RR  'LHS'  $C  'A =D'  $C  'T'
         $(S  $(N  $CK  'TEST'  $RR  'OP_ROOM'  $C  'A =D'
                                       $C  'T'
                   $(FU  $CK  'SHFT'  $RR  'OP_ROOM'  $)N
              $)S
              $CK  'TEST'  $RR  'LHS'  $C  '  = '
                                       $RR  'OP_ROOM'
              $(FU  $R  'FAIL'  $)
         $)S
         $(FU  $CK  'TEST'  $RR  'OP_ROOM'  $C  '  = '
                                       $RR  'LHS'
               $(FU  $CK  'TEST'  $RR  'OP_ROOM'  $C  'T =D'
                                       $C  'C'
                     $(S  $CK  'TEST'  $RR  'OP_ROOM'
                                       $C  'A =D'  $C  'T'
                          $(S  $R  'FAIL'  $)
                      $)FU
                      $CK  'SHFT'  $RR  'OP_ROOM'
              $)FUN
         $)S
         $CK  'SHFT'  $RR  'LHS'
         $CK  'SHFT'  $RR  'OP_ROOM'
    $)N
    $(FU  $CK  'SHFT'  $RR  'LHS'
          $CK  'MOVE'  $RL  'LHS'  $RR  'LHS'
          $(N  $CK  'TEST'  $RR  'OP_ROOM'  $C  'T =D'  $C  ')'
               $(FU  $R  'FAIL'  $)
               $CK  'SHFT'  $RR  'OP_ROOM'
               $CK  'MOVE'  $RL  'OP_ROOM'  $RR  'OP_ROOM'
               $CK  'TEST'  $RL  'LHS'  $C  '  = '
                                       $RL  'OP_ROOM'
               $(FU  $CK  'MOVE'  $RR  'OP_ROOM'
                                       $RL  'OP_ROOM'  $)N
          $)S
```

```
          $CK   'MOVE'   $RR   'LHS'   $RL   'LHS'
          $CK   'MOVE'   $RR   'OP_ROOM'   $RL   'OP_ROOM'
     $)N
     $CK   'TEST'   $RR   'LHS'   $C   'T =D'   $C   '('
     $(S   $RR   'MATCH'   $)
     $CK   'TEST'   $RR   'LHS'   $C   'T =D'   $C   ')'
     $(S   $CK   'TEST'   $RR   'LHS'   $C   'A =D'   $B   '1'
           $(S   $RR   'MATCH'   $)
           $R   'CONTI'
     $)
     $R   'CONTI'
$)X


/***** MAIN PROGRAM FOR TREE MATCHING *****/
$S   'MATCH'   $(XN
     $CK   'TEST'   $RR   'PATTERN'   $C   'T =D'   $C   '('
     $(S   $CK   'COPY'   $RR   'PATTERN'   $RR   'LHS'   $)N
     $CK   'COPY'   $RL   'PATTERN'   $RR   'LHS'
     $R   'ROOT'
     $(N   $CK   'TEST'   $RR   'LHS'   $C   'T =D'   $C   '('
           $(S   $CK   'TEST'   $RR   'LHS'   $C   'A =D'   $B   '0'
                 $(S   $R   'D_DOMINE'   $)N   /** DOMINATION **/
                 $(FU   $CK   'TEST'   $RR   'LHS'   $C   'A =D'
                                               $B   '00001'
                       $(S   $CK   'SHFT'   $RR   'LHS'
                             $CK   'TEST'   $RR   'OP_ROOM'
                                               $C   'T =D'   $C   '('
                             $(S   $CK   'SHFT'   $RR   'OP ROOM'   $)N
                             $R   'CONTI'
                       $)N   /** CONTIGUITY **/
                       $(FU   $R   'ANYONE'   $)N   /** EITHER ONE **/
                 $)N
           $)N
           $(FU   $CK   'COPY'   $C   'SCAN'   $RR   'LHS'
                  $CK   'RSTR'   $RL   'LHS'     /** CHECK RELATION **/
                  $CK   'TEST'   $RL   'LHS'   $C   'A =D'   $B   '0'
                  $(S   $R   'SET_SCAN'   $R   'D_DOMINE'   $)N
                  $(FU   $CK   'TEST'   $RL   'LHS'   $C   'A =D'
                                               $B   '00001'
                       $(S   $R   'SET_SCAN'   $R   'CONTI'   $)N
                       $(FU   $R   'SET SCAN'   $R   'ANYONE'   $)N
                  $)N
           $)N
           $CK   'TEST'   $RR   'LHS'   $C   'A =D'   $B   '1'
     $)S
     $CK   'TEST'   $RR   'OP_ROOM'   $C   'A =D'   $B   '1'
     $(FU   $R   'FAIL'   $)
     $RR   'MAIN'
$)X
```

```
/*****  REFERENCE FOR REPLACEMENT OF DESTINATION ID   *******/
%S   'D_ID'   $'XN
     $CK   'TEST'   $RR   'RHS'   $C   '   = '   $C   '#'
     $(S   $CK   'MOVE'   $RR   'LHS'   $RL   'OP_ROOM'
           $CK   'SHFT'   $RR   'RHS'
           $CK   'TEST'   $RL   'OP_ROOM'   $C   'A =D'   $C   'V'
           $(FU   $CK   'MOVE'   $RR   'OP_ROOM'   $R   'SYS_FREE'   $)N
           $CK   'TEST'   $RR   'LHS'   $C   'T =T'   $RR   'RHS'
           $(FU   $CK   'MOVE'   $RR   'OP_ROOM'   $R   'SYS_FREE'
                  $CK   'MOVE'   $RR   'LHS'   $RL   'OP_ROOM'
           $)N
     $)NS;
     $(F   $CK   'TEST'   $RR   'RHS'   $C   'A =D'   $C   'T'
           $(S   $CK   'MOVE'   $RR   'RHS'   $RL   'OP_ROOM'
                 $CK   'TEST'   $RR   'LHS'   $C   'A =D'   $C   'V'
                 $(S   $CK   'MOVE'   $RR   'LHS'   $R   'SYS_FREE'   $)N
                 $(FU   $CK   'MOVE'   $RR   'LHS'   $R   'SYS_FREE'
                        $CK   'MOVE'   $RR   'OP_ROOM'   $R   'SYS_FREE'
                        $CK   'TEST'   $RR   'LHS'   $C   'T =D'   $C   '('
                 $)FU
           $)N
           $(FU   $CK   'TEST'   $RR   'RHS'   $C   '   = '   $C   '3'
                  $(S   $CK   'SHFT'   $RR   'RHS'
                        $CK   'TEST'   $RR   'LHS'   $C   'A =D'   $C   'C'
                        $(S   $CK   'SHFT'   $RR   'LHS'   $)N
                        $(FU   $CK   'MOVE'   $RR   'LHS'
                                             $R   'SYS_FREE'
                               $CK   'MOVE'   $RR   'OP_ROOM'
                                             $R   'SYS_FREE'
                               $CK   'TEST'   $RR   'LHS'   $C   'T =D'
                                             $C   '('
                        $)FU
                  $)N
                  $(FU   $CK   'MOVE'   $RR   'RHS'   $RL   'OP_ROOM'
                         $CK   'TEST'   $RR   'LHS'   $C   'A =D'
                                             $C   'V'
                         $CK   'SHFT'   $RR   'LHS'
                         $(FU   $CK   'MOVE'   $RR   'OP_ROOM'
                                             $R   'SYS_FREE'   $)N
                         $CK   'TEST'   $RR   'LHS'   $C   'T =T'
                                             $RR   'RHS'
                         $(FU   $CK   'TEST'   $RL   'OP_ROOM'
                                             $C   'A =D'   $C   'V'
                                $(S   $CK   'TEST'   $RR   'LHS'
                                             $C   'T =T'   $RR   'RHS'
                                      $(FU   $CK   'MOVE'   $RR   'LHS'
                                             $R   'SYS_FREE'
                                             $CK 'MOVE' $RR 'OP ROOM'
                                             $R   'SYS_FREE'
                                      $)N
                                $)NS
                                $(FU   $CK   'MOVE'   $RR   'RHS'
                                             $RL   'OP_ROOM'   $)N
```

```
                    $)N
               $)NS
          $)N
     $)N
     $(U  $CK  'TEST'  $RR  'RHS' $C  'T =D'  $C  '('
          $(S  $CK  'MOVE'  $RR  'RHS'  $RL  'OP_ROOM'  $)N
          $CK  'TEST'  $RR  'LHS'  $C  'T =D'  $C  '('
          $(S  $CK  'MOVE'  $RR  'LHS'  $R  'SYS_FREE'  $)N
     $)FUN
     $CK  'TEST'  $RR  'LHS'  $C  'T =D'  $C  'C'
     $(S  $R  'D_TO'  $)
     $CK  'TEST'  $RR  'LHS'  $C  'T =D'  $C  ')'
     $(S  $CK  'TEST'  $RR  'RHS'  $C  'T =D'  $C  '('
          $(S  $CK  'MOVE'  $RR  'RHS'  $RL  'OP_ROOM'
                                         $RR  'TRANS'  $)
               $CK  'SHFT'  $RR  'LHS'
               $CK  'SHFT'  $RR  'RHS'
               $CK  'SHFT'  $RR  'OP_ROOM'
     $)FUN
     $RR  'TRANS'
$)X


/***** REFERENCE FOR REPLACEMENT FOR CONTIGUITY TO      *****/
$S  'C_TO'  $(XN
     $CK  'TEST'  $RR  'LHS'  $C  'T =D'  $C  'C'
$(S  $CK  'TEST'  $RR  'LHS'  $C  'A =D'  $C  'T'
     $(S  $CK  'TEST'  $RR  'OP_ROOM'  $C  'A =D'  $C  'T'
          $(FU  $CK  'SHFT'  $RR  'OP_ROOM'  $)N
     $)S
     $(FU  $CK  'TEST'  $RR  'LHS'  $C  '  = '  $RR  'OP_ROOM'
          $(FU  $CK  'SHFT'  $RR  'OP_ROOM'  $)N
     $)S
$)N
     $CK  'TEST'  $RR  'RHS'  $C  '  = '  $C  '#'
     $(S  $CK  'SHFT'  $RR  'LHS'
          $CK  'SHFT'  $RR  'OP_ROOM'
          $CK  'SHFT'  $RR  'RHS'
          $CK  'TEST'  $RR  'LHS'  $C  'T =T'  $RR  'RHS'
          $(FU  $CK  'MOVE'  $RR  'OP_ROOM'  $R  'SYS_FREE'
                $CK  'MOVE'  $RR  'LHS'  $RL  'OP_ROOM'
          $)N
          $CK  'TEST'  $RR  'LHS'  $C  'T =T'  $RR  'OP ROOM'
          $(FU  $CK  'MOVE'  $RR  'OP_ROOM'  $RL  'OP_ROOM'  $)N
     $)NS
     $(F  $CK  'TEST'  $RR  'RHS'  $C  '  = '  $C  '@'
          $(S  $CK  'SHFT'  $RR  'RHS'
               $(N  $CK  'MOVE'  $RR  'OP_ROOM'  $R  'SYS_FREE'
                    $CK  'TEST'  $RR  'OP_ROOM'  $C  'T =D'
                                          $C  '('
               $)FU
               $(N  $CK  'MOVE'  $RR  'LHS'  $R  'SYS_FREE'
                    $CK  'TEST'  $RR  'LHS'  $C  'T =D'  $C  '('
```

```
            $)FU
        $)N
        $(FU  $CK  'TEST'  $RR  'RHS'  $C  'A =D'  $C  'T'
              $(S  $CK  'MOVE'  $RR  'RHS'  $RL  'OP_ROOM'
                   $CK  'MOVE'  $RR  'LHS'
                                $R  'SYS_FREE'
                   $CK  'MOVE'  $RR  'OP_ROOM'
                                $R  'SYS_FREE'
                   $CK  'TEST'  $RR  'LHS'  $C  'T =1'
                                $RR  'OP_ROOM'
                   $(FU  $CK  'MOVE'  $RR  'OP_ROOM'
                                $RL  'OP_ROOM'  $)N
              $)NS
              $(FU  $CK  'MOVE'  $RR  'RHS'  $RL  'OP_ROOM'
                    $CK  'SHFT'  $RR  'LHS'
                    $CK  'MOVE'  $RR  'OP_ROOM'
                                $R  'SYS_FREE'
                    $CK  'TEST'  $RR  'OP_ROOM'
                                $C  'T =T'  $RR  'LHS'
                    $(FU  $CK  'MOVE'  $RR  'OP_ROOM'
                                $RL  'OP_ROOM'  $)N
              $)NS
        $)N
    $)N
    $(U  $CK  'TEST'  $RR  'RHS'  $C  'T =D'  $C  '('
         $(S  $CK  'MOVE'  $RR  'RHS'  $RL  'OP_ROOM'  $)N
    $)NFU
    $CK  'TEST'  $RR  'LHS'  $C  'T =D'  $C  '('
    $(S  $RR  'TRANS'  $)
    $CK  'TEST'  $RR  'RHS'  $C  'T =D'  $C  '('
    $(S  $CK  'MOVE'  $RR  'RHS'  $RL  'OP_ROOM'
         $R  'C_TO'
    $)
    $CK  'TEST'  $RR  'LHS'  $C  'T =D'  $C  ')'
    $(S  $CK  'SHFT'  $RR  'LHS'
         $CK  'SHFT'  $RR  'RHS'
         $CK  'SHFT'  $RR  'OP_ROOM'
         $RR  'TRANS'
    $)
    $R  'C_TO'
$)X


/***** REFERENCE FOR TREE  REPLACEMENT *****/
$S  'TRANS'  $(XN
         $CK  'SHFT'  $RR  'LHS'
         $CK  'SHFT'  $RR  'RHS'
         $CK  'SHFT'  $RR  'OP_ROOM'
    $CK  'TEST'  $RL  'RHS'  $C  'A =D'  $C  'V'
    $(FU  $CK  'TEST'  $RL  'RHS'  $C  '  = '  $C  '#'
          $(FU  $CK  'MOVE'  $RL  'OP_ROOM'  $R  'SYS_FREE'
                $CK  'MOVE'  $RL  'RHS'  $RL  'OP_ROOM'
          $)N
```

```
    $)NS
    $(N  $CK   'TEST'  $RR   'LHS'  $C   'T =D'  $C   '('
         $(S   $CK   'TEST'  $RR   'LHS'  $C   'A =D'  $B   '0'
               $(S   $CK   'TEST'  $RR   'RHS'  $C   'T =D'  $C   '('
                     $(S   $CK   'SHFT'  $RR   'LHS'
                           $CK   'SHFT'  $RR   'RHS'
                           $CK   'SHFT'  $RR   'OP_ROOM'
                           $R    'D_TO'
                     $)N
                     $(FU  $CK   'MOVE'  $RR   'LHS'
                                         $R    'SYS_FREE'
                           $CK   'MOVE'  $RR   'OP_ROOM'
                                         $R    'SYS_FREE'
                           $R    'D_TO'
                     $)N
               $)N
               $(FU  $CK   'SHFT'  $RR   'LHS'
                     $CK   'SHFT'  $RR   'RHS'
                     $CK   'SHFT'  $RR   'OP_ROOM'
                     $R    'C_TO'
                $)N
         $)N
    $(FU $CK   'COPY'  $C   'SCAN'  $RR   'LHS'
         $CK   'RSTR'  $RL   'LHS'
         $CK   'TEST'  $RL   'LHS'  $C   'A =D'  $B   '0'
         $(S   $(N  $CK   'TEST'  $RR   'LHS'  $C   '  = '
                                         $C   'SCAN'
                    $(FU  $CK   'MOVE'  $RR   'LHS'
                                         $RL   'LHS'  $)N
               $)S
               $CK   'MOVE'  $RR   'LHS'  $R   'SYS_FREE'
               $R    'D_TO'
         $)N
         $(FU $(N  $CK   'TEST'  $RR   'LHS'  $C   '  = '
                                         $C   'SCAN'
                   $(FU  $CK   'MOVE'  $RR   'LHS'  $RL   'LHS'
                                         $)N
               $)S
               $CK   'MOVE'  $RR   'LHS'  $R   'SYS_FREE'
               $R    'C_TO'
         $)N
    $)N
        $CK   'TEST'  $RR   'LHS'  $C   'A =D'  $B   '1'
    $)S
    $RR   'MAIN'
$)X


/***** MAIN PROGRAM FOR TREE TRANSFORMATION *****/
$SN   'MAIN'    $(XN  $CK   'SHFT'  $RR   'TREE'
                      $CK   'SHFT'  $RR   'PATTERN'
                      $CK   'SHFT'  $RR   'LHS'
                      $CK   'SHFT'  $RR   'OP_ROOM'
```

```
                    $CK  'SHFT'  $RR  'CORRESPO'
                    $CK  'SHFT'  $RR  'TEMP'
    $CK  'MOVE'  $RR  'PATTERN'  $RL  'PATTERN'
    $CK  'SHFT'  $RR  'REPLACE'
    $CK  'SHFT'  $RR  'REPLACE'
    $CK  'SHFT'  $RR  'RHS'
    $R  'MATCH'
$CK  'COPY'  $R  'OP_ROOM'  $R  'SYSPRINT'
$CK  'COPY'  $R  'LHS'  $R  'SYSPRINT'
    $CK  'RSTR'  $RL  'LHS'
    $CK  'RSTR'  $RL  'OP_ROOM'
    $CK  'TEST'  $RR  'REPLACE'  $C  'T =D'  $C  '('
    $(S  $CK  'COPY'  $RR  'REPLACE'  $RR  'RHS'  $)N
    $CK  'COPY'  $RL  'REPLACE'  $RR  'RHS'
$CK  'COPY'  $R  'RHS'  $R  'SYSPRINT'
    /** TREE REPLACEMENT **/
    $R  'TRANS'
$CK  'COPY'  $R  'LHS'  $R  'SYSPRINT'
$CK  'COPY'  $R  'RHS'  $R  'SYSPRINT'
$CK  'COPY'  $R  'OP_ROOM'  $R  'SYSPRINT'
    $CK  'RSTR'  $R  'OP_ROOM'
    $CK  'SHFT'  $RR  'OP_ROOM'
/***** EXCHANGE TREE VARIABLES *****/
    $CK  'TEST'  $RL  'CORRESPO'  $C  'A =D'  $B  '1'
    $(S  $CK  'RSTR'  $RR  'OP_ROOM'  $)N
$(FU $CK  'TEST'  $RR  'OP_ROOM'  $C  'A =D'  $C  'V'
    $(S  $CK  'RSTR'  $RL  'CORRESPO'
        $(N  $CK  'TEST'  $RR  'CORRESPO'  $C  '  = '
                                    $RR  'OP_ROOM'
             $(FU  $CK  'MOVE'  $RR  'CORRESPO'
                                    $RL  'CORRESPO'
                   $CK  'TEST'  $RR  'CORRESPO'  $C  'A =D'
                                    $C  'V'
             $)S
        $)S
        $CK  'SHFT'  $RR  'CORRESPO'
        $CK  'MOVE'  $RR  'OP_ROOM'  $R  'SYS_FREE'
        $(N  $CK  'TEST'  $RR  'CORRESPO'  $C  'A =D'  $C  'V'
             $(FU  $CK  'COPY'  $RR  'CORRESPO'
                                    $RL  'OP_ROOM'
                   $CK  'MOVE'  $RR  'CORRESPO'
                                    $RL  'CORRESPO'
             $)N
        $)S
    /** TREE VARIABLE IS NULL **/
        $CK  'TEST'  $RL  'OP_ROOM'  $C  'T =D'  $C  '('
        $(S  $CK  'TEST'  $RR  'OP_ROOM'  $C  'T =D'  $C  ')'
             $(S  $CK  'SHFT'  $RR  'OP_ROOM'
                  $CK  'MOVE'  $RL  'OP_ROOM'  $R  'SYS_FREE'
             $)N
        $)N
        $CK  'SHFT'  $RL  'OP_ROOM'
    $)NFU
```

```
    $CK   'SHFT'   $RR   'OP_ROOM'
    $CK   'TEST'   $RR   'OP_ROOM'   $C   'A =0'   $B   '1'
$)S
    $CK   'MOVE'   $RL   'TREE'   $R   'SYS_FREE'
    $CK   'TEST'   $RR   'TREE'   $C   'T =0'   $C   '('
    $(S   $CK   'MOVE'   $RR   'TREE'   $R   'SYS_FREE'
          $CK   'MOVE'   $RL   'OP_ROOM'   $RR   'TREE'
    $)N
    $CK   'MOVE'   $RL   'OP_ROOM'   $RL   'TREE'
    /**  CLEAR UP WFSS  **/
    $(FUN   $CK   'TEST'   $RL   'LHS'   $C   'A =0'   $B   '1'
            $(FU   $CK   'MOVE'   $RL   'LHS'   $R   'SYS_FREE'   $)N
    $)S
    $(N   $CK   'TEST'   $RL   'RHS'   $C   'A =0'   $B   '1'
          $(FU   $CK   'MOVE'   $RL   'RHS'   $R   'SYS_FREE'   $)N
    $)S
    $CK   'RSTR'   $RL   'CORRESPO'
    $(N   $CK   'TEST'   $RR   'CORRESPO'   $C   '  =  '   $CV   'TO'
          $(FU   $CK   'MOVE'   $RR   'CORRESPO'   $R   'SYS_FREE'
                                                   $)N
    $)S
$CK   'COPY'   $R   'TREE'   $R   'SYSPRINT'
    $R   'RESET'
$)X
```

APPENDIX B

NUCLEOL AND TREE ALGORITHMS

The purpose of this appendix is to introduce readers to NUCLEOL programming by discussing a number of programs for typical tree algorithms -- ranging from simple algorithms which convert one tree representation into another, to the rather elaborate Alpha-Beta pruning technique for the minimal evaluation of game trees.

Familiarity with NUCLEOL to the extent described in [1] is assumed.

ALGORITHMS FOR SEQUENTIAL TREE EXPRESSIONS

Algorithm PE and Algorithm EP (Parenthesized prefix and postfix expressions)

In Chapter 2 we introduced the notion "parenthesized prefix tree expression"; here we introduce a similar concept called "parenthesized postfix (or endorder) tree expression".

Postfix (or endorder) enumeration of nodes for a single tree as well as for an ordered forest may be defined as follows:

To traverse a forest in postfix order: if it is empty, then terminate, otherwise traverse its first tree in postfix order, and thereafter traverse the forest of the remaining trees in postfix order.

To traverse a tree in postfix order: visit the root, and thereafter traverse the forest of its subtrees in postfix order.

A parenthesized postfix expression is obtained by adding a pair of parentheses to enclose all the sons of every nonterminal node.

For example, the parenthesized postfix tree expression of the tree in Figure 1 enumerated by the above process will be

((D E) B (F (I J) G H) C) A

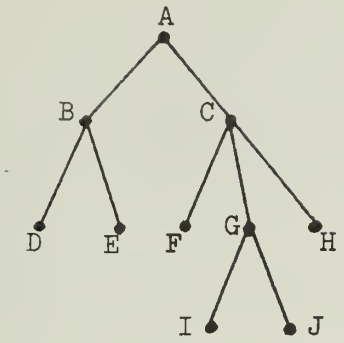


Fig. B-1 A tree

where the corresponding parenthesized prefix tree expression is given by:

A (B (D E) C (F G (I J) H))

The algorithm PE transforms from parenthesized prefix expression to parenthesized postfix expression and the algorithm EP does the converse transformation.

ALGORITHM PE:

```
/***** TREE IN PREFIX ORDER *****/
$S  'TREE'  $(X  $C  'A'  $(  $C  'B'  $(  $C  'D'  $C  'E'
    $)  $C  'C'  $(  $C  'F'  $C  'G'  $C  'H'  $(  $C  'I'
    $)  $)  $)X

/***** TEMPORARY STORAGE *****/
$S  'TEMP'  $(X          $)X


/***** MAIN PROGRAM *****/
$SN  'PRETOEND'  $(XN $CK  'SHFT'  $RR  'TREE'
                      $CK  'SHFT'  $RR  'TEMP'
                      $CK  'SHFT'  $RR  'TREE'
    $(N  $CK  'TEST'  $RR  'TREE'  $C  'T =D'  $C  '('
         $(S  $CK  'MOVE'  $RL  'TREE'  $RR  'TEMP'
              $CK  'MOVE'  $RR  'TREE'  $RL  'TREE'
              $CK  'MOVE'  $RR  'TEMP'  $RR  'TREE'
              $CK  'SHFT'  $RL  'TREE'
              $CK  'RSTR'  $RL  'TREE'
         $)N
         $CK  'SHFT'  $RR  'TREE'
         $CK  'TEST'  $RR  'TREE'  $C  'A =D'  $B  '1'
    $)S
         $CK  'COPY'  $R  'TREE'  $R  'SYSPRINT'
         $R  'SYS_STOP'
    $)X
```

ALGORITHM EP:

```
/***** TREE IN PREFIX ORDER *****/
$S  'TREE'  $(X  $(  $(  $C  'D'  $C  'E'  $)  $C  'B'  $(
$C 'F' $C 'G'     $(  $C  'I'  $)  $C  'H'  $)  $C  'C'  $)
          $C  'A'  $)X

/***** TEMPORARY STORAGE *****/
$S  'TEMP'  $(X            $)X


$SN  'ENDTOPRE'  $(XN $CK  'SHFT'  $RL  'TREE'
                      $CK  'SHFT'  $RR  'TEMP'
                        $CK  'SHFT'  $RL  'TREE'
    $(N  $CK  'TEST'  $RL  'TREE'  $C  'T =D'  $C  ')'
         $(S  $CK  'MOVE'  $RR  'TREE'  $RR  'TEMP'
              $CK  'MOVE'  $RL  'TREE'  $RR  'TREE'
              $CK  'MOVE'  $RR  'TEMP'  $RL  'TREE'
              $CK  'SHFT'  $RR  'TREE'
              $CK  'RSTR'  $RR  'TREE'
         $)N
         $CK  'SHFT'  $RL  'TREE'
         $CK  'TEST'  $RL  'TREE'  $C  'A =D'  $B  '1'
    $)S
         $CK  'COPY'  $R  'TREE'  $R  'SYSPRINT'
         $R  'SYS_STOP'
    $)X
```

Algorithm TL (Generation of parenthesized tree expressions from level sequences)

A sequence of levels of nodes in prefix order gives a unique tree structure (unlabeled). For example, a sequence of the levels of node in prefix order

0 1 2 2 1 2 2 3 3 2

corresponds to the tree structure in Figure B-2, whose parenthesized tree expression is given by

((( ) ( )) (( ) (( ) ( )) ( )))

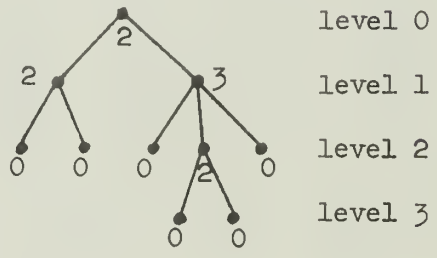


Fig. B-2 Levels and degree of nodes

In general, such a sequence has a form

$$\ell_1 \ell_2 \text{ --- } \ell_i \ell_{i+1} \text{ --- } \ell_n$$

where n is the number of nodes in the tree, and ℓ_i is the level of the i-th node in prefix order. Any two neighboring levels ℓ_i and ℓ_{i+1} satisfy one of the following relations:

1) $\ell_i = \ell_{i+1} - 1$ node i directly dominates node i + 1,

2) $\ell_i = \ell_{i+1}$ nodes i and i + 1 are brothers,

3) $\ell_i > \ell_{i+1}$ all other cases.

The algorithm TL generates a parenthesized tree expression from a given prefix ordered sequence of levels.

ALGORITHM TL:

```
/***** LEVELS IN PREFIX ORDER *****/
 $S  'LEVEL'  $(X  $D '0'  $D  '1'  $D  '2'  $D  '2'
$D  '1' $D  '2'  $D  '1'  $D  '2'  $D  '2'  $D  '2'  $)X

/***** STORAGE FOR THE RESULTING TREE *****/
    $S  'TREE'  $(X     $)X


/***** MAIN PROGRAM *****/
$SN  'PROG'  $(XN $CK  'SHFT'  $RR  'LEVEL'  $CK  'SHFT'
                                      $RR  'LEVEL'
               $CK  'SHFT'  $RR  'TREE'  $CK  'COPY'  $(  $)
                                      $RR  'TREE'
$(N  $(NF  $CK  'TEST'  $RL  'LEVEL'  $C  ' <  '  $RR  'LEVEL'
           $(S  $CK  'SHFT'  $RR  'TREE'
                $CK  'COPY'  $(  $)  $RR  'TREE'
                $CK  'SHFT'  $RR  'LEVEL'
           $)N
     $)F
     $(N  $CK  'TEST'  $RL  'LEVEL'  $C  '  = '  $RR  'LEVEL'
          $(S
          $CK  'TEST'  $RR  'TREE'  $C  'T =D'  $C  '('
          $(S  $CK  'MOVE'  $RR  'TREE'  $RL  'TREE'  $)N
               $CK  'COPY'  $(  $)  $RR  'TREE'
               $CK  'SHFT'  $RR  'LEVEL'
          $)NWF
     $)FU
     $(N  $CK  'TEST'  $RL  'LEVEL'  $C  ' >  '  $RR  'LEVEL'
          $(S  $CK  'MOVE'  $RR  'TREE'  $RL  'TREE'
               $CK  'SHFT'  $RR  'TREE'
               $CK  'SUB'  $RL  'LEVEL'  $D  '1'  $RL  'LEVEL'
          $)NWU
     $)FU
     $CK  'COPY'  $R  'TREE'  $R  'SYSPRINT'
     $CK  'TEST'  $RR  'LEVEL'  $C  'A =D'  $B  '1'
  $)S
     $CK  'COPY'  $R  'TREE'  $R  'SYSPRINT'
    $R  'SYS_STOP'
 $)X
```

Algorithm TD (Generation of parenthesized tree expressions from degree sequences)

A sequence of degrees of nodes in prefix order also gives a unique tree structure. For the tree of Figure B-2 this sequence is given by

2 2 0 0 3 0 2 0 0 0

In general, such a sequence may be expressed as

$$d_1\ d_2\ d_3\ \text{---}\ d_i\ d_{i+1}\ \text{---}\ d_n$$

where d_i is the degree of the i-th node in prefix order ($d_i = 0$ indicates terminal node).

The algorithm TD generates a parenthesized tree expression from a given prefix ordered sequence of degrees of nodes.

ALGORITHM TD:

```
$S 'DEGREES' THIS WFS DESCRIBES TREE IN POLISH PREFIX NOTATION
 $(X $D '3' $D '2' $D '0' $D '0' $D '1' $D '0' $D '3' $D '0'
                                        $D  '0'  $D  '0'  $)X

 $S 'TREE'  WILL CONTAIN THE GIVEN TREE CONVERTED TO
                                        PARENTHESIS NOTATION
$(X THE SMALLEST WFS HAS A SCANNER AND A PAIR OF PARENTHESES .
        MARKED BY AN ATTRIBUTE "X"  $)X


 $SN 'PROG' THE ATTRIBUTE "N"(FOR "NEUTRAL") MARKS THIS THE
                                        EXEC SCANNER
 $(XN    $CK  'SHFT'  $RR  'DEGREES'
         $CK  'SHFT'  $RR  'TREE'
         $CK  'COPY'  $(  $)  THIS IS A IMMEDIATE ARGUMENT
                              $RR  'TREE'
 $(N  BEGINNING OF MAIN LOOP
   $CK  'SHFT'  $RR  'DEGREES'
   $CK  'TEST'   $RL  'DEGREES'  $C  '  = '  $D  '0'
     $(F  $CK  'SHFT'  $RR  'TREE'
          $(N  $CK  'COPY'  $(  $)  $RR  'TREE'
               $CK  'SUB'  $RL  'DEGREES'  $D  '1'
                                        $RL  'DEGREES'
               $CK  'TEST'  $RL  'DEGREES'  $C  '  = '  $D  '0'
          $)S
     $)N
   $(S  $CK  'MOVE'  $RR  'TREE'  $RL  'TREE'
        $(N  $CK  'TEST'  $RR  'TREE'  $C  'T =D'  $C  ')'
             $(S  $CK  'SHFT'  $RR  'TREE'  $)N
          $)F
     $)N
     $CK  'TEST'  $RR  'DEGREES'  $C  'A =D'  $B  '1'
     $)S   END OF MAIN LOOP
     $CK  'COPY'  $R  'TREE'  $R  'SYSPRINT'
     $R  'SYS_STOP'
   $)X
```

Algorithm TA (Lexicographic Binary Tree Generation)

Algorithm TA generates from an arbitrary sequence $A_1, A_2, \ldots, A_n$ of words (character strings) over some ordered alphabet a parenthesized binary tree expression with the following properties:

1) The root is labeled A_1
2) For every node (labeled A_j) its left son (labeled A_i) if it exists, and its right son (labeled A_k) if it exists, the relations

$$A_i < A_j < A_k$$

hold (where "<" means "precedes in lexicographical order").

For example, the sequence of single-character words

H L E B A C F N J K M I

will be converted to:

```
H E B           L J       N
( ( ( A C ) F ) ( ( I K ) ( M ) ) )
```

whose tree is shown in Figure B-3, where nodes labeled "*" have been added to make the resulting tree perfectly binary.

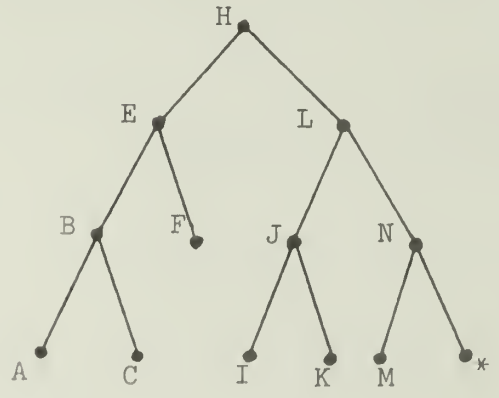


Fig. B-3

In general, sequences consisting of the same words in different order will give rise to different trees.

ALGORITHM TA:

```
 /***** ARBITRARY ALPHABETS *****/
$S  'DATA'  $(X  $C  'G'  $C  'J'  $C  'E'  $C  'B'  $C  'A'
     $C  'C'
 $C  'D'  $C  'F'  $C  'K'  $C  'H'  $C  'I'  $)X

 /***** STORAGE FOR THE RESULT *****/
$S  'TREE'  $(X        $)X


 /***** MAIN PROGRAM *****/
$SN  'PROG'  $(XN    $CK  'SHFT'  $RR  'DATA'
                     $CK  'SHFT'  $RR  'TREE'
                     $CK  'COPY'  $(  $)  $RR  'TREE'
                     $CK  'SHFT'  $RR  'TREE'
                     $CK  'COPY'  $C  '*'  $RR  'TREE'
                     $CK  'COPY'  $C  '*'  $RR  'TREE'
                     $CK  'SHFT'  $RL  'TREE'
                     $CK  'MOVE'  $RR  'DATA'  $RL  'TREE'
                     $CK  'COPY'  $R  'TREE'  $R  'SYSPRINT'
$(N  $CK  'TEST'  $RR  'DATA'  $C  ' < '  $RL  'TREE'
 $CK  'SHFT'  $RR  'TREE'
 $(S  $CK  'TEST'  $RR  'TREE'  $C  ' = '  $C  '*'
 $(F  $CK  'SHFT'  $RR  'TREE'
 $CK  'TEST'  $RR  'DATA'  $C  ' < '  $RL  'TREE'
 $CK  'SHFT'  $RR  'TREE'
 $(F  $CK  'SHFT'  $RR  'TREE'
 $CK  'TEST'  $RR  'TREE'  $C  'T =D'  $C  '('
 $(S  $CK  'MOVE'  $RR  'TREE'  $RL  'TREE'
 $)N
 $)NF
 $)NS
 $)S
 $(F  $CK  'SHFT'  $RR  'TREE'
 $CK  'TEST'  $RR  'TREE'  $C  'T =D'  $C  '('
 $(S  $CK  'MOVE'  $RR  'TREE'  $RL  'TREE'
 $)N
 $(NF  $CK  'TEST'  $RR  'TREE'  $C  ' = '  $C  '*'
 $(F  $CK  'SHFT'  $RR  'TREE'
 $CK  'TEST'  $RR  'DATA'  $C  ' < '  $RL  'TREE'
 $CK  'SHFT'  $RR  'TREE'
 $(F  $CK  'SHFT'  $RR  'TREE'
 $CK  'TEST'  $RR  'TREE'  $C  'T =D'  $C  '('
 $(S  $CK  'MOVE'  $RR  'TREE'  $RL  'TREE'
 $)N
 $)NF
 $)NS
 $)S
 $)N
 $CK  'MOVE'  $RR  'TREE'  $R  'SYS_FREE'
```

```
                    $CK   'COPY'   $(   $)   $RR   'TREE'
                    $CK   'SHFT'   $RR   'TREE'
                    $CK   'COPY'   $C   '*'   $RR   'TREE'
                    $CK   'COPY'   $C   '*'   $RR   'TREE'
                    $CK   'SHFT'   $RL   'TREE'
 $CK   'MOVE'   $RR   'DATA'   $RR   'TREE'
 $CK   'RSTR'   $R    'TREE'
 $CK   'SHFT'   $RR   'TREE'
 $CK   'SHFT'   $RR   'TREE'
 $CK   'COPY'   $R   'TREE'   $R   'SYSPRINT'
 $CK   'TEST'   $RR   'DATA'   $C   'A =D'   $B   '!'
 $)S
 $R   'SYS_STOP'
$)X
```

ALGORITHMS FOR POLISH NOTATION

Polish expressions are sequences of operands (or arguments) and operators (each of which has a degree associated with it, which is the number of operands it requires), formed according to the following rules:

1) An operand is a Polish expression.

2) If $\alpha_1, \alpha_2, \ldots, \alpha_n$ are Polish expressions, and Q is an operator of degree n, then $Q\ \alpha_1\ \alpha_2 \ldots \alpha_n$ is a Polish expression.

We assign ranks to operands and operators as follows:

1) An operand has rank 1.

2) An operator of degree n has rank 1-n.

Let $S : S_1 S_2 \ldots S_n$ be any sequence of operators and operands, and let r_i be the rank of S_i for $i = 1, 2, \ldots, n$. Define the sequence $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ of partial sums as

$$\Sigma_k = \sum_{i=k}^{n} r_i .$$

Then S is a Polish expression if and only if (1) each Σ_i is positive for $i=1, \ldots, n$ and (2) $\Sigma_1 = 1$. In the following examples, operands are denoted by single digits and operators by capital letters. The examples use four operators of degree 2 (A, S, M, D) and one operator of degree 1 (N).

Algorithm TP

This algorithm generates a prefix parenthesized tree expression from a given Polish expression. For example, let a Polish expression be

A S N 1 S 2 3 M 4 D N 5 6 ,

then the algorithm generates

```
A S N   S         M   D N
( ( (1)( 2 3 ) ) ( 4 ( ( 5 ) 6 ) ) )
```

which represents the tree of Figure B-4.

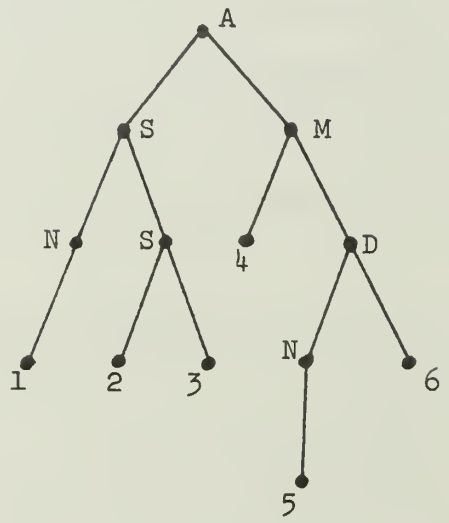


Fig. B-4

ALGORITHM TP:

```
/***** POLISH NOTATION *****/
$S  'POLISH'  $(X  $C  'A'  $C  'S'  $C  'N'  $D  '1'
  $C  'S'  $D  '2'
        $D  '3'  $C  'M'  $D  '4'  $C  'D'  $C  'N'
    $D  '5'  $D  '6'  $)X
/***** OPERATORS TABLE *****/
$S  'OPERATOR'  $(X  $C  'A'  $D  '2'  $C  'S'  $D  '2'
$C  'M'  $D  '2'  $C  'D'  $D  '2'  $C  'N'  $D  '1'  $)X
/***** TREE OF THE FORMULA *****/
$S  'TREE'  $(X  $C  '*'  $)X
/***** TEMPORARY STORAGE *****/
$S  'TEMP'  $(X  $)X

/***** MAIN PROGRAM *****/
$SN  'PROG'  $(XN  $CK  'SHFT'  $RR  'POLISH'
                   $CK  'SHFT'  $RR  'TREE'
                   $CK  'SHFT'  $RR  'OPERATOR'
                   $CK  'SHFT'  $RR  'TEMP'
$(N  $CK  'TEST'  $RR  'POLISH'  $C  'T =D'  $C  'C'
     $(S  $(N  $CK  'TEST'  $RR  'POLISH'  $C  '  = '
                                    $RR  'OPERATOR'
               $(FU  $CK  'SHFT'  $RR  'OPERATOR'  $)N
               $CK  'SHFT'  $RR  'OPERATOR'
          $)S
          $CK  'COPY'  $RR  'OPERATOR'  $RR  'TEMP'
          $CK  'RSTR'  $RL  'OPERATOR'
          $(N  $CK  'TEST'  $RR  'TREE'  $C  '  = '  $C  '*'
               $(FU  $CK  'SHFT'  $RR  'TREE'  $)N
          $)S
          $CK  'MOVE'  $RR  'TREE'  $R  'SYS_FREE'
          $CK  'MOVE'  $RR  'POLISH'  $RL  'TREE'
          $CK  'COPY'  $(  $)  $RR  'TREE'
          $CK  'SHFT'  $RR  'TREE'
          $(N  $CK  'COPY'  $C  '*'  $RR  'TREE'
               $CK  'SUB'  $RR  'TEMP'  $D  '1'  $RR  'TEMP'
               $CK  'TEST'  $RR  'TEMP'  $C  '  = '  $D  '0'
          $)S
     $)N
     $(FU  $(N  $CK  'TEST'  $RR  'TREE'  $C  '  = '  $C  '*'
                $(FU  $CK  'SHFT'  $RR  'TREE'  $)N
           $)S
           $CK  'MOVE'  $RR  'TREE'  $R  'SYS_FREE'
           $CK  'MOVE'  $RR  'POLISH'  $RL  'TREE'
     $)N
     $CK  'COPY'  $R  'TREE'  $R  'SYSPRINT'
     $CK  'TEST'  $RR  'POLISH'  $C  'A =D'  $B  '1'
$)S
     $R  'SYS_STOP'
$)X
```

Algorithm PL

This algorithm computes the level ℓ_i of the i-th element (operator or operand) in a Polish expression. By level we mean the level of the node labeled with this element in the tree corresponding to this expression. Algorithm PL, however, works directly on the Polish expression without converting it into parenthesized form.

The algorithm proceeds in the following two steps.

Step 1: Form the partial sum sequence $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$.

Step 2: Beginning at the left end, form two sequences $\ell_1, \ell_2, \ldots, \ell_n$ and $c_1, c_2, \ldots, c_n$ as follows:

i) $\ell_1 = 1, \quad c_1 = 0$;

ii) for i = 1, 2, ... , n-1

if $\Sigma_i < \Sigma_{i+1}$ then set $\ell_{i+1} = \ell_{1+1}$ and $c_{i+1} = 0$;

if $\Sigma_1 > \Sigma_{i+1}$ then determine the last k which is less than or equal to i such that $c_k = 0$ and then set $\ell_{i+1} = \ell_k$, $c_{i+1} = 1$ and $c_k = 1$.

After termination of the algorithm PL on a Polish expression, ℓ_i is the level of the i-th element in the expression.

ALGORITHM PL:

```
/***** POLISH NOTATION *****/
$S  'POLISH'  $(X  $C  'A'  $C  'S'  $C  'N'  $D  '1'  $C  'S'
$D  '2'  $D  '3'  $C  'M'  $D  '4'  $C  'D'  $C  'N'  $D  '5'
  $D  '6'  $)X

/***** RANKS OF THE NOTATION *****/
$S  'RANKS'  $(X  $)X

/***** PARTIAL SUMS *****/
$S  'P_SUMS'  $(X  $)X

/***** LEVELS OF THE TREE OF THE FORMULA *****/
$S  'LEVEL'  $(X  $)X

/***** C TABLE *****/
$S  'C(I)'  $(X  $)X

/***** OPERATORS TABLE *****/
$S  'OPERATOR'  $(X  $C  'A'  $D  '2'  $C  'S'  $D  '2'
 $C  'M'  $D  '2'  $C  'D'  $D  '2'  $C  'N'  $D  '1'  $)X


/***** MAIN PROGRAM *****/
$SN  'PROG'  $(XN  $CK  'SHFT'  $RR  'POLISH'
                   $CK  'SHFT'  $RR  'RANKS'
                   $CK  'SHFT'  $RR  'P_SUMS'
                   $CK  'SHFT'  $RR  'LEVEL'
                   $CK  'SHFT'  $RR  'C(I)'
                   $CK  'SHFT'  $RR  'OPERATOR'
$(N  $CK  'TEST'  $RR  'POLISH'  $C  'T =D'  $C  'C'
     $(S  $(N  $CK  'TEST'  $RR  'POLISH'  $C  '  = '
                                        $RR  'OPERATOR'
               $CK  'SHFT'  $RR  'OPERATOR'
               $(FU  $CK  'SHFT'  $RR  'OPERATOR'  $)N
          $)S
          $CK  'SUB'  $D  '1'  $RR  'OPERATOR'  $RL  'RANKS'
          $CK  'RSTR'  $RL  'OPERATOR'
     $)N
     $(FU  $CK  'COPY'  $D  '1'  $RL  'RANKS'  $)N
     $CK  'SHFT'  $RR  'POLISH'
     $CK  'TEST'  $RR  'POLISH'  $C  'A =D'  $B  '1'
$)S
$CK  'COPY'  $R  'RANKS'  $R  'SYSPRINT'
     $CK  'SHFT'  $RL  'RANKS'
     $CK  'COPY'  $RR  'RANKS'  $RR  'P_SUMS'
$(N  $CK  'ADD'  $RL  'RANKS'  $RR  'P_SUMS'  $RR  'P_SUMS'
     $CK  'SHFT'  $RL  'RANKS'
     $CK  'TEST'  $RL  'RANKS'  $C  'A =D'  $B  '1'
$)S
```

```
$CK   'COPY'  $R  'P_SUMS'  $R  'SYSPRINT'
      $CK   'TEST'  $RR  'P_SUMS'  $C  ' = '  $D  '1'
      $(F   $CK  'COPY'  $C  'NOT WELL DEFINED'  $R  'SYSPRINT'
            $R  'SYS_STOP'
      $)
      $CK   'COPY'  $D  '1'  $RL  'LEVEL'
      $CK   'COPY'  $D  '0'  $RL  'C(I)'
      $CK   'SHFT'  $RR  'P_SUMS'
$(SN  $CK   'TEST'  $RL  'P_SUMS'  $C  ' < '  $RR  'P_SUMS'
      $(S   $CK   'ADD'  $RL  'LEVEL'  $D  '1'  $RL  'LEVEL'
            $CK   'COPY'  $D  '0'  $RL  'C(I)'
            $CK   'SHFT'  $RR  'P_SUMS'
      $)N
      $CK   'TEST'  $RL  'P_SUMS'  $C  ' = '  $RR  'P_SUMS'
      $(S   $CK   'ADD'  $RL  'LEVEL'  $D  '1'  $RL  'LEVEL'
            $CK   'COPY'  $D  '1'  $RL  'C(I)'
            $CK   'SHFT'  $RR  'P_SUMS'
      $)N
      $CK   'TEST'  $RL  'P_SUMS'  $C  ' > '  $RR  'P_SUMS'
      $(S   $CK   'COPY'  $D  '1'  $RL  'C(I)'
            $(N   $CK   'SHFT'  $RL  'LEVEL'
                  $CK   'SHFT'  $RL  'C(I)'
                  $CK   'TEST'  $RL  'C(I)'  $C  ' = '  $D  '0'
            $)S
            $CK   'COPY'  $RR  'LEVEL'  $RR  'POLISH'
            $CK   'MOVE'  $RL  'C(I)'  $R  'SYS_FREE'
            $CK   'COPY'  $D  '1'  $RL  'C(I)'
            $(N   $CK   'SHFT'  $RR  'LEVEL'
                  $CK   'SHFT'  $RR  'C(I)'
                  $CK   'TEST'  $RR  'C(I)'  $C  'A =D'  $B  '1'
            $)S
            $CK   'MOVE'  $RR  'POLISH'  $RL  'LEVEL'
            $CK   'SHFT'  $RR  'P_SUMS'
      $)N
$CK   'COPY'  $R  'LEVEL'  $R  'SYSPRINT'
      $CK   'TEST'  $RR  'P_SUMS'  $C  'A =D'  $B  '1'
$)S
      $R  'SYS_STOP'
$)X
```

ALGORITHMS FOR GAME TREES

Many problems in artificial intelligence involve the searching of large trees of alternative possibilities. Game playing and theorem proving are examples. This section is concerned with the problem of efficient evaluation of game trees by the "minimax" procedure and its refinement, called "alpha-beta pruning". A further improvement of the latter procedure may be obtained by reordering the nodes of the tree so that more alpha and beta cutoffs are obtained, but this will not be discussed here. The more sophisticated techniques are most useful on very deep trees.

Algorithm MM (MiniMax Procedure)

We assume a tree is given each of whose leaves is labeled with a number. The minimax procedure assigns a number to the root of this tree by repeating the following step as long as possible: if N is a node which has only leaves (L_1, L_2, ... , L_k) as direct descendants, then label N with the number

a) the maximum of the numbers which label L_1, ... , L_k, if the level of N is even, or

b) the minimum of the numbers which label L_1, ... , L_k, if the level of N is odd, then treat N as a new leaf (the program listed will delete L_1, ... , L_k, so it terminates with a tree which consists of the labeled root alone).

ALGORITHM MM:

```
/***** GAME TREE *****/
 $S  'TREE'  $(X  $(  $(  $(  $(  $(  $D  '10'  $)  $(  $D  '8'
    $D  '10'  $D  '17'  $D  '4'  $D  '3'  $)  $)  $D  '4'
       $D  '5'  $)
    $D  '3'  $)  $D  '7'  $D  '4'  $)  $)X

/***** REFERENCE OPERATORS *****/
$S  'OPERATOR'  $(X  $C  ' <= '  $C  ' >= '  $)X

/***** TEMPORARY STORAGE *****/
$S  'TEMP'  $(X       $)X


/***** REFERENCE FOR SETTING OPERATOR *****/
$S  'SETOP'  $(XN  $CK  'TEST'  $RL  'OPERATOR'  $C  'T =D'
                                          $C  '('
    $(S  $CK  'SHFT'  $RR  'OPERATOR'  $)N
    $(F  $CK  'SHFT'  $RL  'OPERATOR'  $)N
       $RR  'PROGRAM'  $)X


/***** MAIN PROGRAM *****/
$SN  'PROGRAM'  $(XN  $CK  'SHFT'  $RR  'OPERATOR'
          $CK 'SHFT' $RR  'TREE'
          $CK  'SHFT'  $RR  'TEMP'
$(N  $CK  'COPY'  $R  'TREE'  $R  'SYSPRINT'
     $CK  'TEST'  $RR  'TREE'  $C  'T =D'  $C  '('
    $(S  $R  'SETOP'
    $CK  'SHFT'  $RR  'TREE'
    $CK  'TEST'  $RR  'TREE'  $C  'T =D'  $C  '('
    $)NF
$CK  'TEST'  $RR  'TREE'  $C  'T =T'  $D  '0'
  $(S  $CK  'SHFT'  $RR  'TREE'
    $CK  'TEST'  $RR  'TREE'  $C  'T =T'  $D  '0'
    $(S  $CK  'TEST'  $RL  'TREE'  $RR  'OPERATOR'
                                    $RR  'TREE'
    $(S  $CK  'MOVE'  $RR  'TREE'  $R  'SYS FREE'
    $)N
    $(F  $CK  'MOVE'  $RL  'TREE'  $R  'SYS FREE'
    $CK  'SHFT'  $RR  'TREE'
    $)N
    $CK  'TEST'  $RR  'TREE'  $C  'T =T'  $D  '0'
    $)NF
  $)NF
$CK  'TEST'  $RR  'TREE'  $C  'A =A'  $RL  'TEMP'
$(F
$CK  'TEST'  $RR  'TREE'  $C  'T =D'  $C  ')'
  $(S  $CK  'MOVE'  $RL  'TREE'  $RR  'TEMP'
    $CK  'SHFT'  $RR  'TREE'
```

```
    $R   'SETOP'
    $CK   'MOVE'   $RI   'TREE'   $R   'SYS_FREE'
    $CK   'MOVE'   $RR   'TEMP'   $RR   'TREE'
    $CK   'TEST'   $RL   'TREE'   $C   'T =T'   $N   '0'
    $(S   $CK   'SHFT'   $RL   'TREE'
    $)N
  $)NF
$)NF
$)S
    $R   'SYS_STOP'
$)X
```

Algorithm AB (Alpha-Beta Procedure)

Alpha-beta is a refinement of the minimax procedure described above, which is more efficient, but is equivalent in the sense that both procedures will always assign the same value to the root of the tree. It saves time by not searching certain branches of the tree. Under certain conditions the value of a branch does not affect the value which is ultimately backed up to higher levels of the tree. Hence, there is no point in evaluating such a branch. When the alpha-beta program detects such a condition, it stops work on one branch and skips to the next. This event is called an alpha or beta cutoff, depending on whether the level of the node where it occurs is even or odd.

Alpha is a number defined by the valves of the successors of a max position (odd levels), and alpha cutoffs occur among successors of a min position (even levels). And beta is established at even levels and generates cutoffs at odd levels. The action of beta cutoffs is exactly the inverse of that for alpha cutoffs.

The effect of alpha and beta cutoffs is to make the tree grow more slowly with depth. Thus the advantage of alpha-beta over the simple minimax procedure increases with the depth of the tree.

ALGORITHM AB:

```
/

/***** GAME TREE *****/
$S  'TREE'  $(X  $(  $D  '6'  $D  '8'  $(  $D  '3'  $(  $D  '8'
 $D  '9'  $)
$)  $D  '10'  $(  $(  $D  '3'  $D  '17'  $D  '6' $) $) $) $)X

/***** REFERENCE OPERATORS *****/
$S  'OPERATOR'  $(X  $C  ' < '  $C  ' > '  $)X

/***** TEMPORARY STORAGE *****/
$S  'TEMP'  $(X      $)X


/***** REFERENCE FOR SETTING OPERATOR *****/
$S  'SETOP'  $(XN  $CK  'TEST'  $RL  'OPERATOR'  $C  'T =D'
                                          $C  '('
    $(S  $CK  'SHFT'  $RR  'OPERATOR'  $)N
    $(F  $CK  'SHFT'  $RL  'OPERATOR'  $)N
       $RR  'PROGRAM'  $)X


/***** MAIN PROGRAM *****/
$S  'MINIMAX'  $(X  $D  '100'  $)X
$SN  'PROGRAM'  $(XN $CK  'SHFT'  $RR  'OPERATOR'
    $CK  'SHFT'  $RR  'TREE'
    $CK  'SHFT'  $RR  'TEMP'
    $CK  'SHFT'  $RR  'MINIMAX'
$(N  $CK  'TEST'  $RR  'TREE'  $C  'T =D'  $C  '('
 $(S  $R  'SETOP'
 $CK  'SHFT'  $RR  'TREE'
$CK  'TEST'  $RR  'TREE'  $C  'T =D'  $C  '('
 $)NF
 $CK  'TEST'  $RR  'TREE'  $C  'T =T'  $D  '0'
 $(S  $CK  'TEST'  $RR  'TREE'  $RR 'OPERATOR'  $RR  'MINIMAX'
 $(S  $CK  'SHFT'  $RL  'TREE'
 $R  'SETOP'
 $CK  'MOVE'  $RR  'TREE'  $R  'SYS_FREE'
$CK  'TEST'  $RL  'TREE'  $C  'T =T'  $D  '0'
$(S
 $CK  'TEST'  $RR  'TREE'  $C  'T =T'  $D  '0'
 $(S  $CK  'TEST'  $RL  'TREE'  $RR 'OPERATOR' $RR  'TREE'
 $(S  $CK  'MOVE'  $RR  'TREE'  $R  'SYS_FREE'
 $)N
 $(F  $CK  'COPY'  $RR  'TREE'  $RR  'MINIMAX'
 $CK  'MOVE'  $RL  'TREE'  $R  'SYS_FREE'
 $CK  'SHFT'  $RR  'TREE'
 $)N
 $CK  'TEST'  $RR  'TREE'  $C  'T =T'  $D  '0'
 $)NF
 $(F  $CK  'COPY'  $RL  'TREE'  $RR  'MINIMAX'
```

```
 $)N


$)N
$)NF
 $(F  $CK  'SHFT'  $RR  'TREE'
 $CK  'TEST'  $RR  'TREE'  $C  'T =T'  $D  '0'
 $(S  $CK  'TEST'  $RL  'TREE'  $RR 'OPERATOR'  $RR  'IREE'
 $(S  $CK  'MOVE'  $RR  'TREE'  $R  'SYS_FREE'
 $)N
$(F
 $CK  'MOVE'  $RL  'TREE'  $R  'SYS_FREE'
 $CK  'SHFT'  $RR  'TREE'
 $)N
 $CK  'TEST'  $RR  'TREE'  $C  'T =T'  $D  '0'
 $)NF
 $(F  $CK  'COPY'  $RL  'TREE'  $RR  'MINIMAX'
 $)N
 $)N
 $)N
 $CK  'TEST'  $RR  'TREE'  $C  'A =A'  $RL  'TEMP'
 $(F  $CK  'TEST'  $RR  'TREE'  $C  'T =D'  $C  ')'
 $(S  $CK  'MOVE'  $RL  'TREE'  $RR  'TEMP'
 $CK  'SHFT'  $RR  'TREE'
 $R  'SETOP'
 $CK  'MOVE'  $RL  'TREE'  $R  'SYS_FREE'
 $CK  'MOVE'  $RR  'TEMP'  $RR  'TREE'
 $CK  'TEST'  $RL  'TREE'  $C  'T =T'  $D  '0'
 $(S  $CK  'SHFT'  $RL  'TREE'
 $)N
 $)NF
 $)NF
 $CK  'COPY'  $R  'TREE'  $R  'SYSPRINT'
 $CK  'COPY'  $RR  'MINIMAX'  $R  'SYSPRINT'
 $)S
 $R  'SYS_STOP'
 $)X
```

BOUND BY
HNM

UNIVERSITY OF ILLINOIS-URBANA
510.84 IL6R no. C002 no.374-378(1970
Optimum network design using NOR-OR gate
3 0112 088399065